分组密码 uBlock 详细设计

吴文玲　张　蕾

郑雅菲　李灵琛

中国科学院软件研究所

2019 年 10 月

摘　要

本报告首先详细介绍分组密码 uBlock 算法，然后简要介绍 uBlock 的设计原理，初步的安全性分析评估，以及在各种平台的实现性能等。uBlock 是一族分组密码算法，分组长度和密钥长度支持 128 和 256 比特，分别记为 uBlock-128/128、uBlock-128/256 和 uBlock-256/256。uBlock 算法的整体结构、S 盒、扩散矩阵、密钥扩展等设计，处处体现了安全、实现效率以及适应性的平衡。uBlock 算法对差分分析、线性分析、积分分析、不可能差分分析、中间相遇攻击等分组密码分析方法具有足够的安全冗余。uBlock 算法充分考虑了现代微处理器的计算资源，可以利用 SSE 和 AVX2 等指令集高效实现。uBlock 算法的硬件实现简单而有效，既可以高速实现，保障高性能环境的安全应用，也可以轻量化实现，满足资源受限环境的安全需求。

目　录

1. 引言

为响应中国密码学会举办的全国密码算法设计竞赛，我们研制完成了分组密码 uBlock 算法，本报告将给出算法描述，算法的设计理念，初步的安全性分析以及实现性能评估。

uBlock 算法的设计目标是安全性高、可扩展性好、适应性强的分组密码，以满足多个行业领域对分组密码算法的应用需求。主要技术指标参照竞赛要求，具体如下：

1）分组长度和密钥长度至少支持 128、256 比特可选；

2）能够抗差分分析、不可能差分、线性分析、积分攻击和相关密钥攻击等分组密码分析方法，同时应有一定的安全冗余；

3）具有一定的灵活性，适合多种软硬件环境实现，实现效率与国际主流同类算法相当。

2. uBlock 分组密码

uBlock 的分组长度为 128 或 256 比特，密钥长度为 128 或 256 比特，记为 uBlock-128/128，uBlock-128/256，uBlock-256/256，它们的迭代轮数 r 分别为 16，24，24。

2.1 符号

在 uBlock 算法中，使用了如下符号：

X	n 比特明文
Y	n 比特密文
K	k 比特密钥
RK^i	n 比特轮密钥
$PL_n, PR_n, PL_n^{-1}, PR_n^{-1}$	$n/16$ 个字节的向量置换
s	4 比特 S 盒
S_n, S_n^{-1}	$n/8$ 个 s 盒的并置
S_k	$k/16$ 个 s 盒的并置
PK_1	16 个半字节的向量置换
PK_2, PK_3	32 个半字节的向量置换
$\oplus$	模 2 加运算
$<<<b$	循环左移 b 比特
$<<<_{32}b$	分块 32 比特循环左移 b 比特
$\|\|$	表示比特串的连接

2.2 加密算法

加密算法由 r 轮迭代变换组成，轮变换如图 1 所示。输入 n 比特明文 X 和轮密钥 $RK^0, RK^1, ..., RK^r$，输出 n 比特密文 Y。加密算法 E 如下：

$uBlock.Enc(X, RK^0, RK^1, ..., RK^r)$

$X_0 \| X_1 \leftarrow X$

for $i = 0$ *to* r-1 *do*

$RK_0^i \| RK_1^i \leftarrow RK^i$

$X_0 \leftarrow S_n(X_0 \oplus RK_0^i)$

$X_1 \leftarrow S_n(X_1 \oplus RK_1^i)$

$X_1 \leftarrow X_1 \oplus X_0$

$X_0 \leftarrow X_0 \oplus (X_1 <<<_{32} 4)$

$X_1 \leftarrow X_1 \oplus (X_0 <<<_{32} 8)$

$X_0 \leftarrow X_0 \oplus (X_1 <<<_{32} 8)$

$X_1 \leftarrow X_1 \oplus (X_0 <<<_{32} 20)$

$X_0 \leftarrow X_0 \oplus X_1$

$X_0 \leftarrow PL_n(X_0)$

$X_1 \leftarrow PR_n(X_1)$

$Y \leftarrow RK^r \oplus (X_0 \| X_1)$

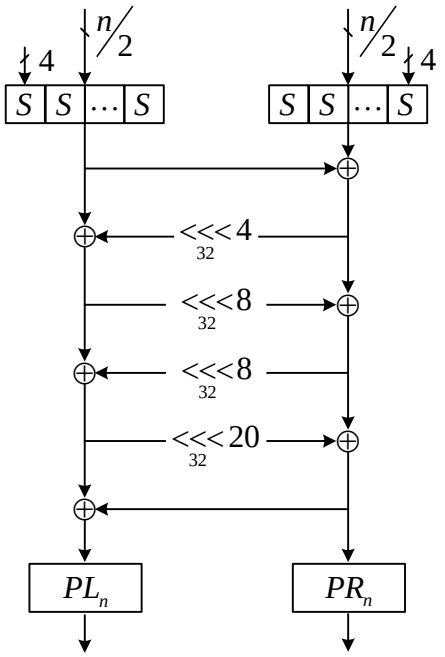

图 1. 加密轮变换

其中基本模块定义如下：

（1） S_n

S_n 由 $n/8$ 个相同的 4 比特 s 盒并置而成，定义如下：

$$S_n : (\{0,1\}^4)^{n/8} \to (\{0,1\}^4)^{n/8}$$
$$(x_0, x_1, ..., x_{n/8-1}) \to (s(x_0), s(x_1), ..., s(x_{n/8-1}))$$

4 比特 s 盒如表 1 所示。

表 1. s

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$s(x)$	7	4	9	c	b	a	d	8	f	e	1	6	0	3	2	5

（2） PL_n 和 PR_n

PL_n 和 PR_n 都是 $m(=n/16)$ 个字节的向量置换，具体见表 2。比如 PL_{128} 的表达式为：

$$PL_{128} : (\{0,1\}^8)^8 \to (\{0,1\}^8)^8$$
$$(y_0, y_1, y_2, ..., y_6, y_7) \to (z_0, z_1, z_2, ..., z_6, z_7)$$
$$z_0 = y_1, \quad z_1 = y_3, \quad z_2 = y_4, \quad z_3 = y_6,$$
$$z_4 = y_0, \quad z_5 = y_2, \quad z_6 = y_7, \quad z_7 = y_5.$$

表 2. PL_n 和 PR_n

PL_{128}	{1,3,4,6,0,2,7,5}
PR_{128}	{2,7,5,0,1,6,4,3}
PL_{256}	{2,7,8,13,3,6,9,12,1,4,15,10,14,11,5,0}
PR_{256}	{6,11,1,12,9,4,2,15,7,0,13,10,14,3,8,5}

2.3 解密算法

解密算法由 r 轮迭代变换组成，输入 n 比特密文 Y，轮密钥 $RK^r, RK^{r-1}, ..., RK^0$，输出 n 比特明文 X。解密算法如下：

$uBlock.Dec(Y, RK^r, RK^{r-1}, ..., RK^0)$
$Y_0 \| Y_1 \leftarrow Y$
$for \quad i = r \quad to \quad 1 \quad do$
$RK_0^i \| RK_1^i \leftarrow RK^i$
$Y_0 \leftarrow Y_0 \oplus RK_0^i$
$Y_1 \leftarrow Y_1 \oplus RK_1^i$
$Y_0 \leftarrow PL_n^{-1}(Y_0)$
$Y_1 \leftarrow PR_n^{-1}(Y_1)$
$Y_0 \leftarrow Y_0 \oplus Y_1$
$Y_1 \leftarrow Y_1 \oplus (Y_0 <<<_{32} 20)$
$Y_0 \leftarrow Y_0 \oplus (Y_1 <<<_{32} 8)$
$Y_1 \leftarrow Y_1 \oplus (Y_0 <<<_{32} 8)$
$Y_0 \leftarrow Y_0 \oplus (Y_1 <<<_{32} 4)$
$Y_1 \leftarrow Y_1 \oplus Y_0$
$Y_0 \leftarrow S_n^{-1}(Y_0)$
$Y_1 \leftarrow S_n^{-1}(Y_1)$
$X \leftarrow RK^0 \oplus (Y_0 \| Y_1)$

其中 S_n^{-1}、PL_n^{-1} 和 PR_n^{-1} 分别是 S_n、PL_n 和 PR_n 的逆，具体见表 3 和表 4.

表 3. s^{-1}

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$s^{-1}(x)$	c	a	e	d	1	f	B	0	7	2	5	4	3	6	9	8

表 4. PL_n^{-1} 和 PR_n^{-1}

PL_{128}^{-1}	{4,0,5,1,2,7,3,6}
PR_{128}^{-1}	{3,4,0,7,6,2,5,1}
PL_{256}^{-1}	{15,8,0,4,9,14,5,1,2,6,11,13,7,3,12,10}
PR_{256}^{-1}	{9,2,6,13,5,15,0,8,14,4,11,1,3,10,12,7}

2.4 密钥扩展算法

将 k 比特密钥 K 放置在 k 比特寄存器，取寄存器的左 n 比特作为轮密钥 RK^0，然后, 对 $i=1,2,...,r$，更新寄存器, 并取寄存器的左 n 比特作为轮密钥 RK^i。

寄存器更新方式如下：

$$
\begin{array}{|l|}
\hline
K_0 \| K_1 \| K_2 \| K_3 \leftarrow K \\
K_0 \| K_1 \leftarrow PK_t(K_0 \| K_1) \\
K_2 \leftarrow K_2 \oplus S_k(K_0 \oplus RC_i) \\
K_3 \leftarrow K_3 \oplus T_k(K_1) \\
K \leftarrow K_2 \| K_3 \| K_1 \| K_0 \\
\hline
\end{array}
$$

其中 S_k 是 $k/16$ 个 4 比特 s 盒的并置， T_k 是对 K_1 的每半字节 $\otimes 2$，有限域 $GF(2^4)$ 的不可约多项式 $m(x)=x^4+x+1$；RC_i 为 32 比特常数，作用在 K_0 的左 32 比特。PK_t 有三种情况，$t=1,2,3$，PK_1、PK_2 和 PK_3 分别用于 uBlock-128/128、uBlock-128/256 和 uBlock-256/256 的密钥扩展算法。PK_1 是 16 个半字节的向量置换， PK_2 和 PK_3 都是 32 个半字节的向量置换，具体见下表。

表 5. PK_t

PK_1	{6,0,8,13,1,15,5,10,4,9,12,2,11,3,7,14}
PK_2	{10,5,15,0,2,7,8,13,14,6,4,12,1,3,11,9,24,25,26,27,28,29,30,31,16,17,18,19,20,21,22,23}
PK_3	{10,5,15,0,2,7,8,13,1,14,4,12,9,11,3,6,24,25,26,27,28,29,30,31,16,17,18,19,20,21,22,23}

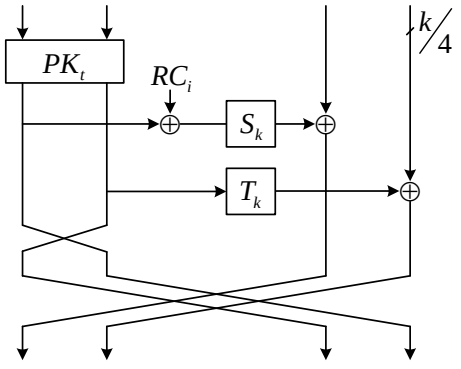

图 2. 密钥状态更新函数

32 比特常数 RC_i 由 8 级 LFSR 生成，初始条件为 $c_0 = c_3 = c_6 = c_7 = 0$，$c_1 = c_2 = c_4 = c_5 = 1$；对 $i \geq 8$，$c_i = c_{i-2} \oplus c_{i-3} \oplus c_{i-7} \oplus c_{i-8}$。令

$$a_i = c_i\overline{c}_{i+1}c_{i+2}c_{i+3}c_{i+4}c_{i+5}c_{i+6}c_{i+7},\ a_i' = c_i\overline{c}_{i+1}c_{i+2}\overline{c}_{i+3}c_{i+4}\overline{c}_{i+5}c_{i+6}c_{i+7},$$

$$a_i'' = c_ic_{i+1}c_{i+2}\overline{c}_{i+3}c_{i+4}c_{i+5}c_{i+6}\overline{c}_{i+7},\ a_i''' = c_ic_{i+1}c_{i+2}c_{i+3}c_{i+4}\overline{c}_{i+5}c_{i+6}\overline{c}_{i+7},$$

则 $RC_i = a_i \| a_i' \| a_i'' \| a_i'''$。

常数 $RC_i (i = 1, 2, ..., 24)$ 的 16 进制表示如下表。

表 6. 常数 RC_i

RC_1	988cc9dd	RC_{13}	dcc88d99
RC_2	f0e4a1b5	RC_{14}	786c293d
RC_3	21357064	RC_{15}	30246175
RC_4	8397d2c6	RC_{16}	a1b5f0e4
RC_5	c7d39682	RC_{17}	8296d3c7
RC_6	4f5b1e0a	RC_{18}	c5d19480
RC_7	5e4a0f1b	RC_{19}	4a5e1b0f
RC_8	7c682d39	RC_{20}	55410410
RC_9	392d687c	RC_{21}	6b7f3a2e
RC_{10}	b3a7e2f6	RC_{22}	17034652
RC_{11}	a7b3f6e2	RC_{23}	effbbeaa
RC_{12}	8e9adfcb	RC_{24}	1f0b4e5a

3. uBlock 的设计原理

3.1 整体结构

uBlock 算法的整体结构称为 PX 结构，如图 3 所示。PX 结构是 SP 结构的一种细化结构，PX 是 Pshufb-Xor 的缩写，Pshufb 和 Xor 分别是向量置换和异或运算指令。采用 S 盒和分支数的理念，PX 结构针对差分分析和线性分析具有可证明的安全性，对于不可能差分分析、积分分析、中间相遇攻击等分析方法具有相对成熟的分析评估理论支持。在同等安全的条件下，PX 结构具有更好的软件和硬件实现性能。利用 SSE/AVX 指令集提供的 128/256 比特寄存器的异或运算和向量置换指令，对于由 4 比特 S 盒构造的非线性变换层，以及线性变换中基于 4 比特的向量置换均仅需一条指令即可实现；因此，该加密方法的软件实现中每轮变换仅需要 m 条异或指令和 $m+4$ 条向量置换指令。此外，该实现方法不需要查表操作，不仅可以提供高性能软件实现，还可抵抗缓存计时等侧信道攻击。该结构采用 4 比特 S 盒构造非线性变换层，相较于现有分组密码标准 AES 等算法采用的 8 比特 S 盒，更易于硬件实现，延迟、面积、能耗等硬件实现指标更优。其次，4 比特 S 盒在各种平台的软件实现灵活，可以采用查表方式，也可以采用比特切片的实现方式。此外，该结构非常灵活，根据 $P_j(j=1,...,m)$ 、 P_{m+1} 和 P_{m+2} 等不同的参数选择都有相应的轮函数与之对应，便于设计多参数规模、安全性高且实现代价低的分组密码算法族。$P_j(j=1,...,m)$ 可以选择面向字的置换，也可以选择分块循环移位，设计者可以灵活选取分块大小 a 和移位数 $b_j(j=1,...,m)$ 。

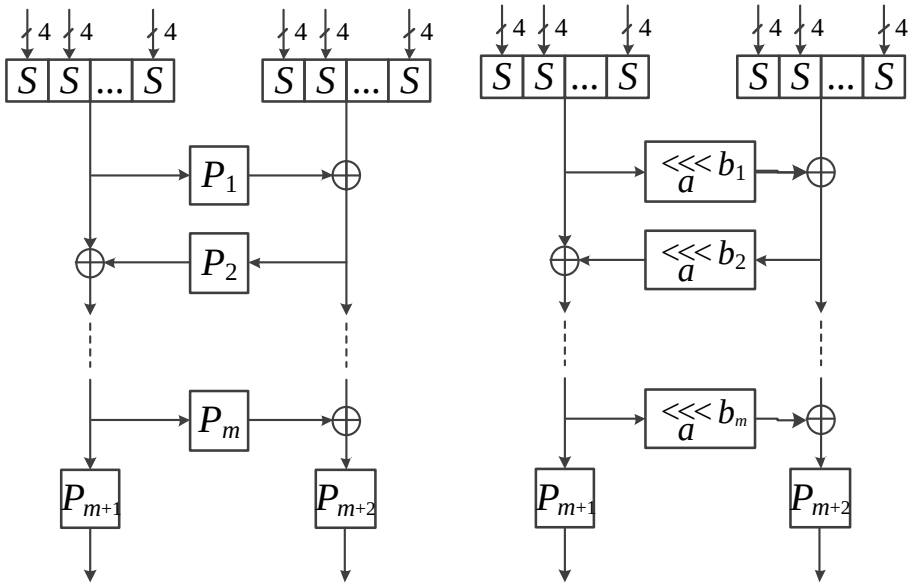

图 3. PX 整体结构

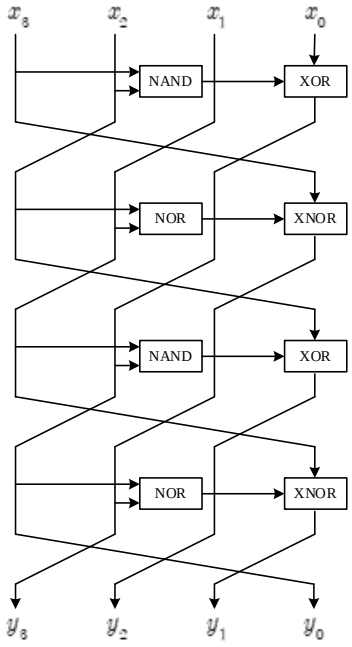

图 4. S 盒硬件实现流程

3.2 S 盒

uBlock算法采用4比特S盒，一个原因是4比特S盒可以用SSE等指令集快速实现，另一个原因是硬件实现代价的考虑。和8比特S盒相比，4比特S盒具有明显的

硬件优势,不仅硬件实现代价小，而且延迟能耗等方面都有明显优势。

uBlock算法的S盒如图4所示，需要2个与非门，2个或非门，2个异或门，2个异或非门，关键路径为4，每比特平均1个乘法电路，针对侧信道的TI实现可以3share、分解深度2，属于最优情况。在安全性方面，uBlock算法的S盒没有不动点、最大线性概率达到最佳2^{-2}、最大差分概率达到最佳2^{-2}、代数次数达到最佳3。最后，uBlock算法中加密和密钥扩展使用同一个4比特S盒，可以实现最大程度的硬件电路模块复用，有效降低硬件实现规模和代价。

3.3 扩散层

uBlock 算法的扩散层由两部分组成，一个是 PX 结构（图 3）中的 $P_j(j=1,...,m)$ 及异或运算，记为线性变换 B，另一个是 PX 结构（图 3）中的 P_{m+1} 和 P_{m+2}。考虑到算法的软件实现性能和适应性，$P_j(j=1,...,m)$ 选择为分块 32 的循环移位，而且期望 m 尽可能小，b_j 尽可能为 0 或者是 8 的倍数。m 的选取和分块大小 a 有关，也和扩散层的分支数有关。分块大小为 32，采用 4 比特 S 盒，利用 Feistel 结构构造 16*16 的二元域最优扩散层，最少需迭代 6 次，即 m=6. 依据我们关于二元域最优扩散层的研究成果[GWG15]，选取 $b_1=b_6=0$，$b_2=4$，$b_3=b_4=8$，$b_5=20$，保证 16*16 的二元域矩阵分支数为 8，达到二元域上的最佳。

P_7 和 P_8 对应算法中的 PL_n 和 PR_n，它们都是面向字节的向量置换，具有最佳的硬件性能，而且适宜 8 位处理器。线性变换 B 和 PL_n 以及 PR_n 的联合使用，使得 uBlock 算法的整体结构对于差分分析和线性分析等分析方法具有可证明安全性。经过反复分析测试，我们选定目前采用的 PL_n 和 PR_n，uBlock-128 和 uBlock-256 的整体结构的全扩散轮数分别为 2 和 3，差分和线性活动 S 盒的个数、不可能差分路径、积分区分器等达到设计要求。

3.4 密钥扩展算法

密钥扩展算法采用随用即生成的方式生成轮密钥，不增加存贮负担；状态更新函数简单且可逆，使得密钥扩展算法的实现更加灵活。密钥扩展算法采用与加密算法不同的整体结构，利用向量置换构造了扩展 Feistel 结构，相比广义 Feistel 结构扩散更快，相比 Feistel 结构更轻量。安全性方面主要考虑的是双

系攻击和相关密钥类攻击的抵抗力，密钥扩展算法的扩散性，轮密钥和种子密钥的关系。实现性能方面，充分利用向量置换，尽量不增加实现成本，为资源受限环境的应用提供保障。向量置换的设计主要考虑密钥扩展算法的扩散性和相关密钥活跃 S 盒的个数等因素，其次兼顾硬件和软件实现性能。S 盒的个数是安全性和实现性能的折中结果。⊗2 操作以很小的代价提供半字节内部的扩散，硬件实现⊗2 操作仅需 1 比特异或。同时，向量置换、非线性 S 盒与状态字间的异或操作的联合使用，提供了足够的非线性和良好的扩散效果，能够使得算法抵抗双系攻击等分析方法。

4. uBlock 的安全性

本部分初步评估 uBlock 算法针对典型分组密码分析方法的安全性。为了方便，uBlock-256/256 简写为 uBlock-256，uBlock-128 指分组长度为 128 比特的 uBlock。

4.1 差分分析

针对差分分析[BS93]，采用搜索差分路径活跃 S 盒个数的方法评估算法抵抗攻击的能力。该方法也被其它大量算法的安全性评估所采用，如 Camellia[AIK01]、CLEFIA[SSA07]、LBlock[WZ11]等。我们通过计算机搜索了 uBlock-128 算法的差分活跃 S 盒个数，结果见表 7。10 轮 uBlock-128 算法至少有 66 个差分活跃 S 盒，由于算法采用的 S 盒的最大差分概率为 2^{-2}，因此 10 轮 uBlock-128 算法的最大差分路径概率满足 $DCP_{\max}^{10r} \le 2^{66\times(-2)} = 2^{-132}$，说明 10 轮 uBlock-128 已经不存在差分分析可利用的有效差分路径。考虑到 uBlock-128 的迭代轮数和全扩散轮数，可以相信全轮 uBlock-128 算法针对差分分析是安全的。

表 7. uBlock-128 的活跃 S 盒数

轮数	1	2	3	4	5	6	7	8	9	**10**
活跃 S 盒数	1	8	13	24	30	36	43	50	56	**66**

对于 uBlock-256，利用超级 S 盒和分支数，我们证明了 4 轮 uBlock-256 至少有 32 个差分活跃 S 盒。因此，16 轮 uBlock-256 至少有 128 个差分活跃 S 盒，说明 uBlock-256 最长存在 15 轮的有效差分路径。进一步，考虑到 uBlock-256 的全扩散轮数为 3，迭代轮数为 24，因此，全轮 uBlock-256 算法针对差分分析是安全的。

4.2 线性分析

针对线性分析[M94]，我们也采用类似的搜索活跃 S 盒个数的方法来评估该算法抵抗攻击的能力。对于 uBlock-128 算法，线性活跃 S 盒个数的测试结果同上小节的差分测试结果。对于 uBlock-256 算法，可以证明 4 轮至少有 32 个线性活跃 S 盒。由于 uBlock 算法使用的 S 盒的最大线性概率为 2^{-2}，因此，考虑到 uBlock 算法的迭代轮数和全扩散轮数，可以相信 uBlock 算法针对线性分析是安全的。

4.3 不可能差分分析

不可能差分分析[BBS04]是非常有效的分组密码分析方法之一，不可能差分分析的关键一步是构造不可能差分区分器。我们利用差分传播系统搜索 uBlock 的不可能差分区分器[WM12]，结果显示 uBlock-128 存在 4 轮不可能差分，uBlock-256 存在 5 轮不可能差分。

记一个 r-轮分组密码算法 E 的差分传播系统为 Φ，ΔP 和 ΔC 分别是 E 的输入差分和输出差分，$\Phi|_{\Delta P,\Delta C}$ 表示将 ΔP 和 ΔC 带入 Φ 之后得到的差分传播系统。如果差分传播系统 $\Phi|_{\Delta P,\Delta C}$ 无解，则 $\Delta P \to \Delta C$ 是 E 的一条 r-轮不可能差分，记做 $\Delta P \not\to_r \Delta C$。从给定的输入和输出差分出发，利用差分传播系统推导中间变量的信息，这些信息可以通过完全求解线性方程以及部分求解非线性方程组成的子方程系统得到。进一步，新得到的信息又可以反馈到差分传播系统，得到更多的信息。如此反复，直到发现矛盾或者不能再求解出新的信息为止。在推导信息的过程中，如果发现矛盾，则可知给定的输入输出差分是一条不可能差分。

对基于半字节的 uBlock 算法，以半字节为单位引入变量，即所有变量都取值于有限域 $GF(2^4)$，选择输入和输出差分的汉明重量为 1，搜索最长轮数的不可能差分。

对于 uBlock-128 算法,构造了 1024 条 4 轮不可能差分，具体如下:

序号	输入差分	输出差分
1	10000000000000000000000000000000	10000000000000000000000000000000
2	10000000000000000000000000000000	01000000000000000000000000000000
3	10000000000000000000000000000000	00100000000000000000000000000000
…	…	…

1023	00000000000000000000000000000001	00000000000000000000000000000010
1024	00000000000000000000000000000001	00000000000000000000000000000001

对于 uBlock-256 算法,构造了 4096 条 5 轮不可能差分，具体如下:

序号	输入差分	输出差分
1	10000000000000000000000000000000 00000000000000000000000000000000	10000000000000000000000000000000 00000000000000000000000000000000
2	10000000000000000000000000000000 00000000000000000000000000000000	01000000000000000000000000000000 00000000000000000000000000000000
3	10000000000000000000000000000000 00000000000000000000000000000000	00100000000000000000000000000000 00000000000000000000000000000000
…	…	…
4095	00000000000000000000000000000000 00000000000000000000000000000001	00000000000000000000000000000000 00000000000000000000000000000010
4096	00000000000000000000000000000000 00000000000000000000000000000001	00000000000000000000000000000000 00000000000000000000000000000001

利用上述不可能差分区分器，可以给出缩减轮 uBlock 的不可能差分分析；但是考虑到 uBlock 算法的扩散性和迭代轮数，可以相信全轮 uBlock 针对不可能差分分析提供了足够的安全冗余。

4.4 积分分析

积分分析[DKR97]起源于针对 Square 算法的 Square 攻击。Square 攻击被提出之后，由其扩展得到的一系列攻击方法，如：饱和攻击、多子集攻击等相继出现。2002 年，Knudsen[KW02]等将这些攻击方法统一归纳为积分攻击。2015 年，Todo[YT15]提出可分性质的概念，它是一种一般化的积分性质，利用可分性可以搜索分组密码的积分区分器。

uBlock 算法整体结构为 SP，S 盒的代数次数为 3，因此，uBlock-128 是 (4,3,32)-SPN, uBlock-256 是(4,3,64)-SPN。依据 Todo 的搜索结果可知，uBlock-128 存在 7 轮积分区分器,数据复杂度为2^{124}选择明文；uBlock-256 存在 8 轮积分区分器,数据复杂度为2^{252}选择明文。具体如下:

对于uBlock-128算法,任意选取一个半字节取常数,其余31个半字节活跃，

7 轮 uBlock-128 加密之后， 2^{124} 个密文的异或和为全 0。比如:

序号	输入	输出异或和
1	caaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	00000000000000000000000000000000
2	acaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	00000000000000000000000000000000
…	…	…
32	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac	00000000000000000000000000000000

其中 c 表示常数,a 表示活跃.

对于uBlock-256算法,任意选取一个半字节取常数,其余63个半字节活跃，8 轮 uBlock-256 加密之后， 2^{252} 个密文的异或和为全 0。比如:

序号	输入	输出异或和
1	caaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	00000000000000000000000000000000 00000000000000000000000000000000
2	acaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	00000000000000000000000000000000 00000000000000000000000000000000
…	…	…
64	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac	00000000000000000000000000000000 00000000000000000000000000000000

文献[ZW15]针对可分性传播过程中计算量急增的事实,提出了一种提前删除冗余向量的技术。文献[YT16]将可分性扩展到比特可分性,对可分性的刻画更准确。文献[XZB16]基于 MILP 给出了更有效的积分区分器搜索算法。利用这些新技术有可能给出 uBlock 更好的积分区分器;但是考虑到 uBlock 算法的扩散性和迭代轮数，可以相信 uBlock 针对积分分析提供了足够的安全冗余。

4.5 中间相遇攻击

中间相遇攻击的主要思想是将加密过程划分为前向和后向两个部分,在计算连接处的中间状态过程中两部分都不覆盖所有的密钥比特,借助一定的存储可以给出优于穷举的恢复密钥算法。近几年，中间相遇攻击得到了快速发展，提出了三子集中间相遇攻击、δ-集中间相遇攻击等分析技术[AMB13][JJIY14][LW17]。δ-集中间相遇攻击，通过选择满足特殊性质的明文结构，使某个中间值可表示为明文与部分未知常数比特的函数，预先建表，通过部分加解密与预先建立的表进行匹配来筛选密钥。我们评估了 uBlock 算法抵抗此类中间相遇攻击的能力，

结果显示 6 轮 uBlock-128/128、8 轮 uBlock-128/256、6 轮 uBlock-256/256 存在此类中间相遇攻击，攻击过程中的主要参数以及攻击结果汇总如下表。

表 8. uBlock 算法的中间相遇攻击

算法名称	攻击轮数	区分器长度	数据	时间	存储
uBlock-128/128	6	3	2^{60}	$2^{125.42}$	$2^{103.9}$
uBlock-128/256	8	4	2^{64}	2^{245}	$2^{238.99}$
uBlock-256/256	6	3	2^{64}	$2^{117.42}$	$2^{89.99}$

6 轮 uBlock-128/128 的中间相遇攻击

本小节的δ-集为 16 个状态的集合，这 16 个状态仅在一个 nibble 处互不相同且取遍所有可能值，在其它位置取相同值。

令 uBlock-128/128 的第 0 个 nibble 是活跃的，可得如下性质：

16 个消息$\{P^0,...,P^{15}\}$经过 3 轮 uBlock-128/128 加密以及一次密钥加后输出的 30 个 4 比特差分

$$\{Y_0^4[0]\oplus Y_1^4[0],\cdots,Y_0^4[0]\oplus Y_{15}^4[0],Y_0^4[17]\oplus Y_1^4[17],\cdots,Y_0^4[17]\oplus Y_{15}^4[17]\}$$

可完全由 26 个 4 比特状态决定：

$$Y^0[0],Y^1[0,2,9,10,11,17,22,23,24,25,31],$$
$$Y^2[0,1,2,3,4,5,6,16,17,18,19,20,21,23]$$

其中$Y[t]=X[t]\oplus RK[t]$。

上述过程可参见图 5，其中红色表示受明文差分影响的状态 nibble，蓝色表示影响$Y^4[0]$和$Y^4[17]$处差分、同时受明文差分影响的状态 nibble。30 个 4 比特差分共有$2^{4\times 26}$种可能值。

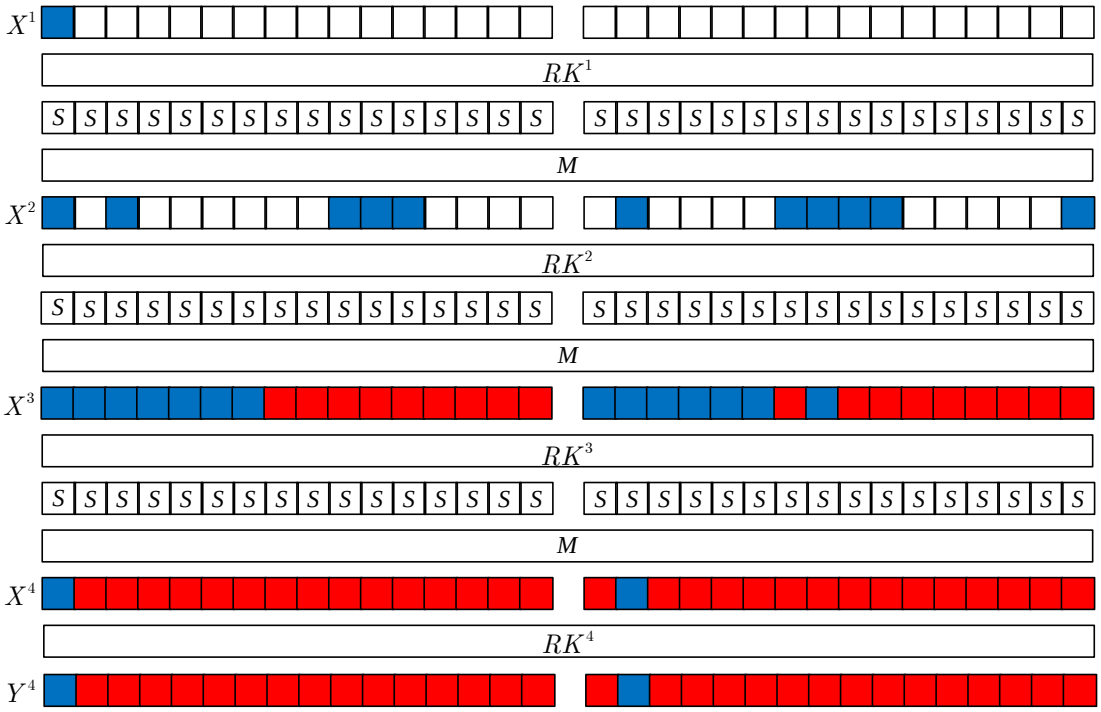

图 5. uBlock-128/128 的 3 轮区分器

基于上述性质，在区分器前加 1 轮，后加 2 轮(第 5 轮密钥 RK^4 已包含在区分器内)给出 6 轮 uBlock-128/128 的中间相遇攻击。攻击过程概述如下：

离线阶段：计算 26×4 比特差分的所有 2^{104} 种可能值，并存储于表 T。

在线阶段：

1. 选择明文 P_0，猜测密钥 $RK^0[0,1,4,5,6,16,17,18,19,21,23]$，部分加密 P_0 得到区分器输入端 $X^1[0]$。
2. 在区分器输入端构造 δ-集，令 $X^1[0]$ 遍历所有可能 16 种可能值，具体为令 $X^1[0]$ 异或 15 种非零差分。其他位置为未知常值。
3. 猜测密钥 $RK^0[0,2,3,4,7,17,18,19,20,21,23]$，部分解密 δ-集，即可得到明文输入的活跃差分，与 P_0 异或可得到符合 3 轮区分器特性的、包含 16 个明文的集合。
4. 询问加密 oracle，得到明文集合相应密文；
5. 猜测 $RK^5[0,1,2,3,8,9,10,16,17,22,23,25,30,31]$ 和 RK^6，解密密文得到 $Y^4[0]$ 和 $Y^4[17]$；
6. 检查所得序列是否存在表 T 中。

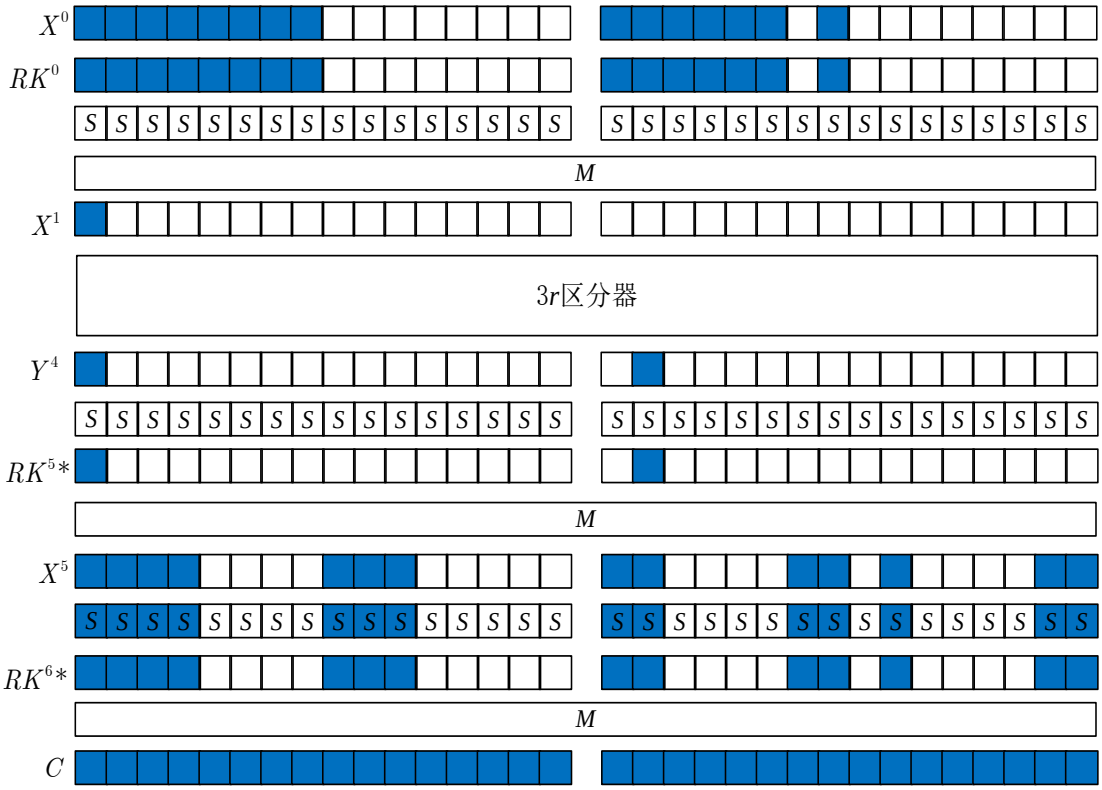

图 6.6 轮 uBlock-128/128 的中间相遇攻击

部分加密和解密阶段分别涉及 15 和 46 个 nibble；利用等价密钥的概念将最后的两个轮密钥加操作通过线性变换的逆，移至 M 层前、S 盒层后的位置，则通过猜测等价密钥可降低 30 个 nibble 的猜测密钥量。攻击共需要猜测 31 个 nibble 的密钥信息（如图 6 所示）。筛选概率为 $2^{104-15\times 8}=2^{-16}$，一次筛选后剩余密钥量为 2^{108}。攻击的数据复杂度为 2^{60} 选择明文；计算复杂度约为 $2^{124}\times 2^4$ 次部分加解密，约为 $2^{125.42}$ 次 6 轮 uBlock-128/128 加密；存储复杂度为 2^{104} 个 $15\times 8=120$ 比特序列，约为 $2^{103.9}$ 个 128 比特块。

表 9.6 轮 uBlock-128/128 的中间相遇攻击参数

区分器轮数	δ-集位置	存储差分序列位置	区分器变量个数	前端轮数	后端轮数
3	$X^1[0]$	$Y^4[0,17]$	26	1	2

8 轮 uBlock-128/256 的中间相遇攻击

表 10.8 轮 uBlock-128/256 的中间相遇攻击参数

区分器轮数	δ-集位置	存储差分序列位置	区分器变量个数	前端轮数	后端轮数
4	$X^1[0,17]$	$Y^5[0]$	59	1	3

利用等价密钥，在线阶段需要猜测的密钥量为 60 个 nibble。筛选概率为 $2^{(59-255)\times 4}=2^{-784}$，一次筛选后剩余密钥量为远小于 1，可认为仅有正确的猜测密钥通过筛选。攻击的数据复杂度为 2^{64} 选择明文；计算复杂度约为 $2^{240}\times 2^{8}$ 次部分加解密，即 2^{245} 次 8 轮 uBlock-128/256 加密；存储复杂度为 2^{236} 个 $255\times 4=1020$ 比特序列，约为 $2^{238.99}$ 个 128 比特块。

6 轮 uBlock-256/256 的中间相遇攻击

表 11. 6 轮 uBlock-256/256 的中间相遇攻击参数

区分器轮数	δ-集位置	存储差分序列位置	活跃变量个数	前端轮数	后端轮数
3	$X^1[1,36]$	$X^4[35]$	22	1	2

利用等价密钥，在线阶段需要猜测的密钥量为 28 个 nibble。筛选概率为 $2^{(22-255)\times 4}=2^{-932}$，一次筛选后剩余密钥量为 $2^{112}\times 2^{-932}$，远小于 1，即仅有正确的猜测密钥可以通过筛选。攻击的数据复杂度为 2^{64} 选择明文；计算复杂度约为 $2^{112}\times 2^{8}$ 次 部分加解密，即 $2^{117.42}$ 次 6 轮 uBlock-256/256 加密；存储复杂度为 2^{88} 个 $255\times 4=1020$ 比特序列，约为 $2^{89.99}$ 个 256 比特块。

4.6 双系攻击

双系攻击[BKR11]是 2011 年由 Bogdanov 等人提出，并给出了全轮 AES 在单密钥下的第一个分析结果。双系攻击的基本流程：首先选取适当的活跃密钥将密钥空

间进行分类划分；然后在密钥划分的基础上，建立最长轮的双系结构；接下来选取合适的匹配位置，通过验证中间状态是否匹配来删减错误密钥；最后穷举验证所有的候选密钥，找出正确的密钥。双系攻击需要遍历整个密钥空间，所以目前被认为是优化的穷举攻击。我们评估了 uBlock 算法抵抗双系攻击的能力，结果显示 11 轮 uBlock-128/128、16 轮 uBlock-128/256 以及 13 轮 uBlock-256/256 存在双系攻击；全轮 uBlock 算法不存在有效的双系攻击。

11 轮 uBlock-128/128 的双系攻击

（1）攻击过程

1．密钥空间的划分

选择主密钥状态经过一轮迭代之后的状态作为等价主密钥。

将 2^{128} 的主密钥划分为 2^{112} 个集合，每个集合中含有 2^{16} 种取值。在一个密钥集合中， 第 15、20 个 nibble 位置以及第 16、22 个 nibble 位置共 16 比特的值是变化的，其它比特是固定不变的。

将一个集合中的 2^{16} 个密钥区分如下：

$$K[i,j]_{\{15,20\}} = i \qquad K[i,j]_{\{16,22\}} = j$$

在给定密钥划分的情形下，可以唯一地确定两种（i 或 j）密钥活跃模式下各轮密钥的活跃情形如图 7 所示，红色块代表只受到 i 密钥活跃影响的轮子密钥位置，蓝色块代表只受到 j 密钥活跃影响的轮子密钥位置，紫色块代表同时受到两密钥活跃影响的轮子密钥位置。

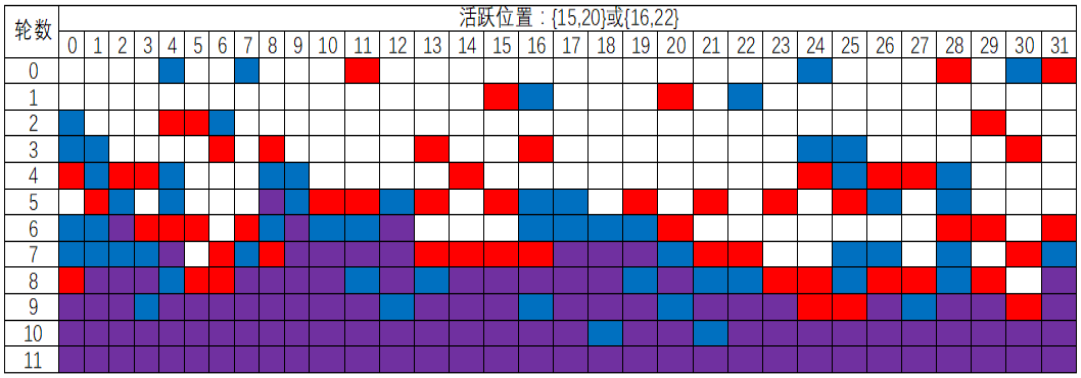

图 7. 11 轮 uBlock-128/128 算法双系分析的密钥划分

2．双系的建立

双系分析的关键部分之一就是建立双系结构。在上述密钥划分的情形下，构成的相关密钥差分路径在前 2 轮可以建立 8 维的双系结构，具体扩散路径如图 8 所示：

2.1　对于给定的 P_0，在 $K[0,0]$ 下加密计算得到 S_0。

2.2　对 P_0，在 $K[i,0]$ 下加密计算的中间状态记为 S_i。

2.3　对 S_0，在 $K[0,j]$ 下解密得到的明文记为 P_j。

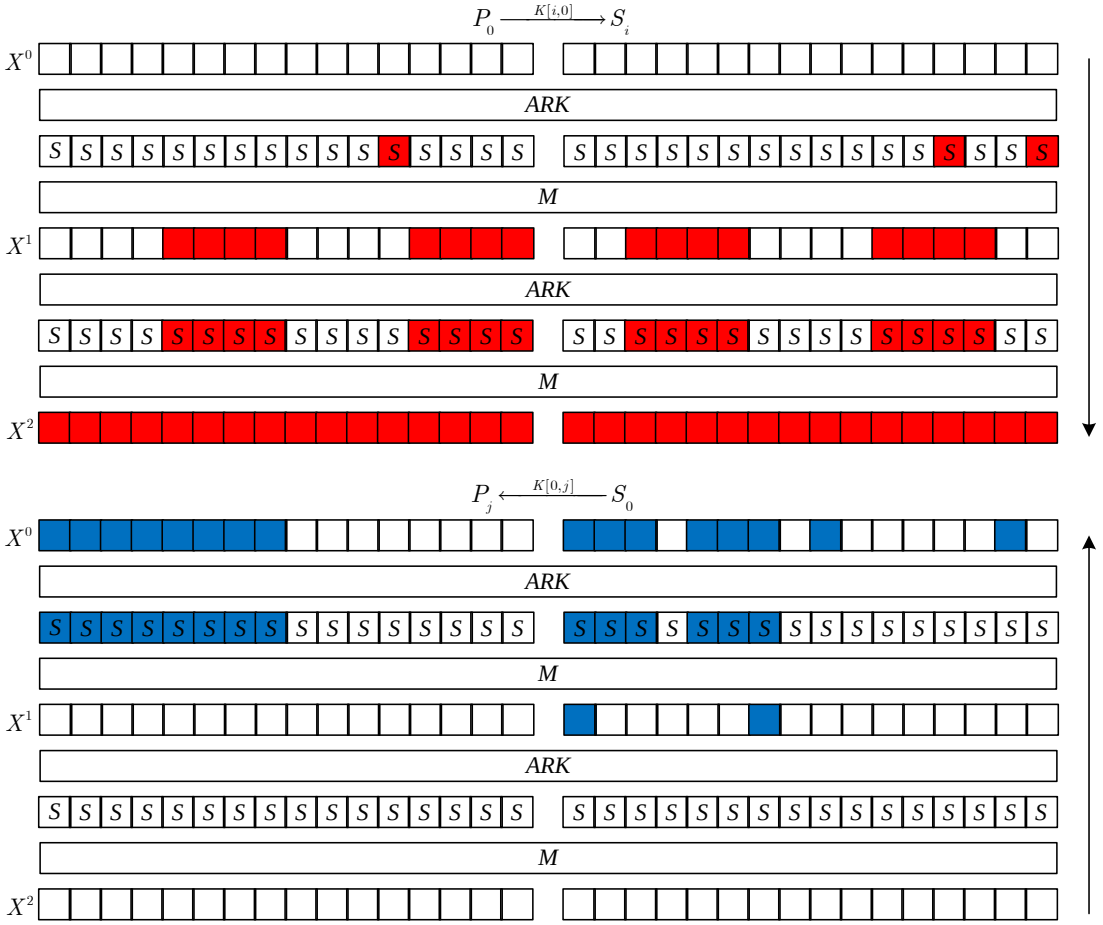

图 8.　uBlock-128/128 算法 2 轮的双系结构

因为前向和后向的差分传播路径不含共同的非线性活跃部分，因此满足任意 P_j，在 $K[i,j]$ 下加密计算得到 S_i。这样就成功地建立起 2 轮的双系结构，此过程中 2.1 步需要计算 64 个 S 盒，2.2 步需要计算 19 个 S 盒 2^8 次，2.3 步需要计算 14 个 S 盒 2^8 次。

3.　密文获取

通过询问加密 Oracle，得到相应明文对应的密文值。

4. 中间匹配

如果测试密钥中的一个$K[i,j]$是秘密密钥K_{secret}，那么它将中间状态S_i映射到密文C_j。这样，我们需要验证是否$\exists i,j:S_i \xrightarrow{K[i,j]} C_j$。为了降低整个攻击的时间复杂度，此过程采用部分匹配的技巧，匹配位置在第 6 轮的第 0、1、2、23 共四个 nibble 位置。

前向匹配：对于每一个固定的S_i，在不同的密钥$K[i,j]$下计算得到匹配位置的过程如图 9 上所示。其中蓝色部分代表此部分的计算受到活跃密钥 j 的影响，需要计算2^8次；黄色部分表示此部分的计算不受活跃密钥 j 的影响，只需要计算一次；白色部分表示部分匹配过程中此部分不需要计算。统计可知，对每一个S_i，需要计算2^8次的 S 盒共有 64 个，需要计算 1 次的 S 盒共有 47 个。

后向匹配：对于每一个固定的C_j在不同的密钥$K[i,j]$下计算得匹配位置的过程如图 9 下所示；其中红色部分代表此部分的计算受到活跃密钥i的影响，需要计算 2^8 次；白色部分表示部分匹配过程中此部分不需要计算。统计可知，对每一个C_j，需要计算2^8次的 S 盒共有 116 个。

5. 穷搜剩余密钥

对一个密钥集合来说，共有2^{16}个密钥，匹配位置共有 16 比特信息，因此在一个密钥集合中平均剩余 1 个密钥作为正确密钥候选，共剩余 2^{112} 个候选密钥。穷搜所有的候选密钥，直至找到正确密钥。

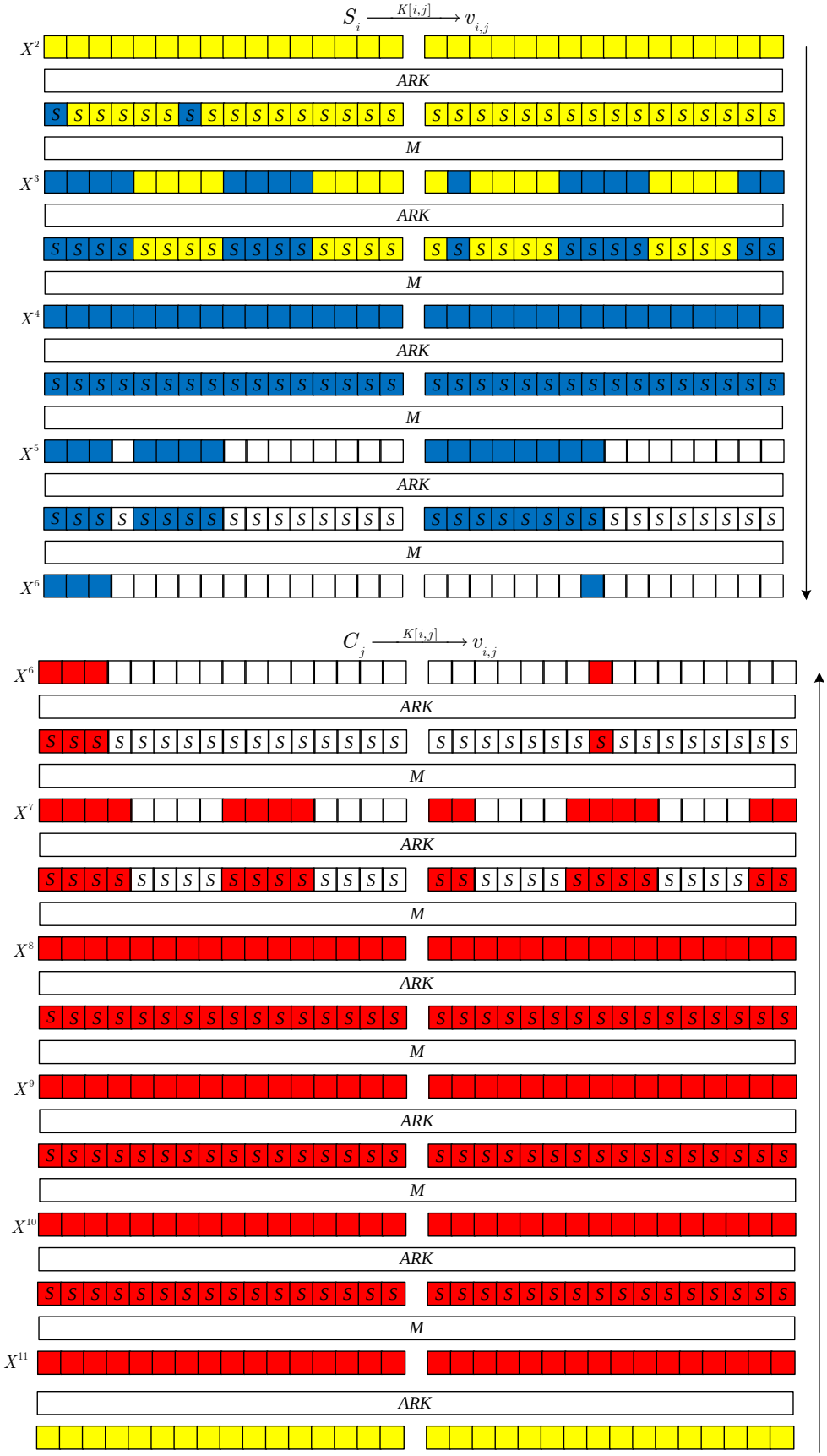

图 9. uBlock-128/128 算法的部分匹配过程

（2）复杂度分析

表 12.　uBlock-128/128 算法双系分析参数

轮数	BicLen	BicDim	活跃密钥	匹配位置	前向轮数	后向轮数
11	2r(0-1)	8×8	{15,20} {16,22}	$X^6[0,1,2,23]$	4	5

数据复杂度：密钥恢复过程中的数据复杂度是由询问加密 Oracle 的明文个数来决定的。因此，所有的密钥群建立双系的过程中产生的所有的互不相同的明文的个数是整个攻击过程中的数据复杂度。对于所有密钥测试集合，我们都固定 P_0的值，在双系建立过程中，反向解密导致明文位置有 16 个 nibble 活跃，因此整个攻击过程需要的数量不超过 2^{64}个选择明文。

时间复杂度：时间复杂度的衡量单位为 11 轮的 uBlock-128/128 算法的加密过程，11 轮加密过程中每轮含有 32 个 S 盒的计算；此外，密钥扩展算法中需要迭代生成第 2 轮至第 11 轮的轮子密钥，以及加密结束之后的白化密钥，共计 12 个轮子密钥，每个轮密钥生成需要 8 个 S 盒。因此，衡量基准单位为 11*（32+8）=440 个 S 盒操作。

时间复杂度主要包含三个方面，双系结构的建立，中间相遇过程以及穷搜密钥过程。

对于每一个密钥集合：

$$C_{biclique} = 64 + 2^8 \times 33 = 8512$$

$$C_{MITM} = 2^8(47 + 2^8 \times (64+116)) = 11808512$$

密钥扩展算法中，使用预计算存储的方式，需要计算 2^8次的 S 盒共有 34 个，需要计算 2^{16}次的 S 盒有 21 个，因此$C_{key} = 88 + 2^8 \times 34 + 2^{16} \times 21 = 1385048$。

除此之外，每一个密钥集合最后剩余 1 个候选密钥。

整个攻击过程中的时间复杂度计算如下：

$$\begin{aligned} C &= 2^{n-2d}\left(\frac{C_{Biclique} + C_{MITM} + C_{Key}}{440} + 1\right) \\ &= 2^{112}\left(\frac{8512 + 11808512 + 1385048}{440} + 1\right) \\ &= 2^{126.873} \end{aligned}$$

16 轮 uBlock-128/256 的双系攻击

选择主密钥状态经过两轮迭代之后的状态作为等价主密钥，可以给出 16 轮 uBlock-128/256 的双系攻击，数据复杂度为 2^{92} 选择明文，时间复杂度为 $2^{254.989}$ 。

表 13. 16 轮 uBlock-128/256 双系攻击参数

轮数	BicLen	BicDim	活跃密钥	匹配位置	前向轮数	后向轮数
16	3r(0-2)	8×8	{42,48} {46,57}	$X^8[0,1,2,23]$	5	8

13 轮 uBlock256/256 的双系攻击

选择主密钥状态经过一轮迭代之后的状态作为等价主密钥，可以给出 13 轮 uBlock-256/256 的双系攻击，数据复杂度为 $2^{23*4=92}$，时间复杂度为 $2^{254.899}$。

表 14. 13 轮 uBlock-256/256 双系攻击参数

轮数	BicLen	BicDim	活跃密钥	匹配位置	前向轮数	后向轮数
13	2r(0-1)	8×8	{37,48} {57,63}	$X^7[0,1,36,37]$	5	6

5. uBlock 的实现性能

uBlock 算法的设计过程中，充分兼顾了算法在 8/32/64 等软件环境和硬件平台的实现性能。uBlock 算法的主要运算仅包含 4 比特 S 盒、以 4 比特/8 比特为单位的置换、以及异或等逻辑运算，非常适合现代 CPU 支持的 SIMD 指令集实现。并行结构、简单的 S 盒和线性变换，使得 uBlock 算法的硬件实现简单而有效，既可以高速实现，也可以轻量化实现。

5.1 PC 软件性能

对于 uBlock 算法在 32/64 位平台上的实现性能，可以采用传统的查表方式实现，也可以利用现代 CPU 支持的 SIMD 指令集实现，比如 Intel CPU 支持的 SSE 等指令集，可以支持 128 比特寄存器和相应的逻辑运算、向量置换等。考虑到 uBlock 算法的主要运算仅包含 4 比特 S 盒、以 4 比特/8 比特为单位的置换、以及异或等逻辑运算，非常适合用向量置换指令。

我们实际测试了 uBlock 算法在软件平台上的实现速度。测试环境为：CPU Intel Core i7-3740QM CPU @ 2.70GHz，内存 16.0 GB，操作系统 Windows 7 专业版 64 位，编译环境 Microsoft Visual Studio 2010。随机生成明文和密钥，测试运行 100 次，取平均速度，速度单位 Mbps。uBlock 算法各版本针对不同长度的消息（包括短消息和长消息）的软件实现速度检测结果如下表所示。

表 15. uBlock 算法的软件实现性能

PC 上的软件实现性能		256 Bytes		1 MB	
		Cycle/Byte	Mbps	Cycle/Byte	Mbps
加密速度	uBlock-128/128	13.1	1641	11.8	1809
	uBlock-128/256	21.1	1056	17.4	1216
	uBlock-256/256	17.1	1268	13.6	1566
解密速度	uBlock-128/128	13.0	1670	11.3	1869
	uBlock-128/256	20.1	1098	16.7	1289
	uBlock-256/256	16.8	1276	13.4	1571

根据测试结果，在长消息下采用 SSE 指令集实现的 uBlock-128/128 算法的加密速度可以达到 11.8 Cycles/Byte（1809 Mbps），解密速度可以达到 11.3 Cycles/Byte（1869 Mbps）；uBlock-256/256 算法的加密速度可以达到 13.6 Cycles / Byte（1566 Mbps），解密 速度可以达到 13.4 Cycles/Byte（1571 Mbps）。进一步，如果采用最新版本 CPU 支持的 AVX2 指令，uBlock-256/256 算法的实现速度有提升，而且随着 AVX 指令集的更新换代，以及寄存器长度的扩展，uBlock-256/256 算法的性能优势将会展现。

5.2 ARM 软件性能

对于 uBlock 算法在 32 位 ARM 平台上的实现性能，与 64 位 Windows 平台类似，大部分 ARM 平台处理器也提供了扩展指令集——neon 指令集，支持相应的 SIMD 运算，包括向量置换、128 比特逻辑运算等。基于此可以给出 uBlock 算法在 ARM 平台下基于 neon 指令集的优化实现。此外，对于嵌入式微处理器中使用的低功耗 ARM，计算能力和 RAM/ROM 等资源可能受限，此时可以直接按照算法描述给出 uBlock 算法的基础实现，不再采用扩展指令集、查大表等优化实现技术。

我们实际测试了 uBlock 算法在两种 32bit ARM 软件平台上的优化实现和基础实现的加/解密速度。测试环境分别为：（1）ARM Cortex A53，主频 1.4 GHz；（2）Ateml ATSAM3X8E ARM Cortex M3，RAM 96KB，ROM 1MB，Flash 512KB，主频 84 MHz。随机生成明文和密钥，测试运行 100 次，取平均速度，速度单位 Mbps。uBlock 算法各版本针对不同长度的消息（包括短消息和长消息）的软件实现速度检测结果如下表所示。

表 16. uBlock 算法的 ARM 平台软件实现性能

优化实现（ARM Cortex A53）		256 Bytes	1 MB
加密速度	uBlock-128/128	66.1 Mbps	168 Mbps
	uBlock-128/256	48.8 Mbps	127 Mbps
	uBlock-256/256	38.6 Mbps	103.9 Mbps
解密速度	uBlock-128/128	60.2 Mbps	163.2 Mbps
	uBlock-128/256	45.5 Mbps	121.2 Mbps
	uBlock-256/256	35.3 Mbps	93.1 Mbps

基础实现（ARM Cortex M3）		256 Bytes
加密速度	uBlock-128/128	1.26 Mbps
	uBlock-128/256	0.77 Mbps
	uBlock-256/256	0.56 Mbps
解密速度	uBlock-128/128	1.26 Mbps
	uBlock-128/256	0.82 Mbps
	uBlock-256/256	0.69 Mbps

5.3 硬件性能

为说明 uBlock 算法的硬件实现性能，我们利用 Verilog 语言实现了 uBlock，与算法设计一致，采用以轮为单位的迭代实现，测试了硬件仿真实现的性能。测试环境为：仿真工具 ModelSim SE，综合工具 Sysnopsys Design Compiler，工艺元件库为 TSMC 90nm tcbn90ghpwc_ccs。

采用 32 个分组数据进行加/解密，只调用 1 次密钥扩展算法。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）。速率单位 Mbps。

表 17. uBlock 算法的硬件实现性能

硬件实现性能	128/128	128/256	256/256
运算周期	576	832	832
时钟约束	1.38 ns	1.32 ns	1.25 ns
加密速率	5152.9 Mbps	3729.6 Mbps	7876.9 Mbps
解密速率	4996.8 Mbps	3616.6 Mbps	7638.2 Mbps
面积	45767 um2	54555 um2	86992 um2
等效面积	16216 GE	19329 GE	30822 GE
加密吞面比	0.113	0.068	0.091
解密吞面比	0.109	0.066	0.088

此外，考虑到部分工作模式中不需要解密运算，我们也测试了 uBlock 算法在单独加密模式下的硬件实现性能。采用 以轮为单位的迭代实现，包含加密和密钥扩展算法。类似的，采用 32 个分组数据进行加密，只调用 1 次密钥扩展算法。算法实现的加密运算用时包含密钥扩展运算用时（单位：微秒）。算法电路面积规模包含加密、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）。速率单位 Mbps。

表 18. uBlock 算法的加密硬件实现性能

加密硬件实现性能	128/128	128/256	256/256
运算周期	576	832	832
时钟约束	1.25 ns	1.20 ns	1.20 ns
加密速率	5688.9 Mbps	4102.6 Mbps	8205.1 Mbps
面积	24517 um2	25729 um2	41062 um2
等效面积	8687 GE	9116 GE	14549 GE
加密吞面比	0.232	0.159	0.200

5.4 其它平台和应用场景

uBlock算法适宜轻量化实现，可适用于资源受限环境。uBlock算法设计采用了4比特S盒作为非线性模块，充分考虑了轻量化硬件实现的成本。和常用的8比特S盒相比，4比特S盒具有明显的硬件实现优势，不仅硬件实现代价小，而且延迟、能耗等方面都有明显优势。uBlock算法的S盒仅需要2个与非门，2个或非门，2个异或门，2个异或非门，关键路径为4。uBlock算法的线性层采用了简单的循环移位、异或、字节置换运算，具有很好的硬件实现性能。密钥扩展采用随用即生成的方式生成轮密钥，不增加存贮的负担；常数利用8比特线性反馈移位寄存器产生，硬件实现代价小。此外，加密和密钥扩展使用同一个4比特S盒，可以实现最大程度的硬件电路模块复用，有效降低硬件实现规模和代价。

uBlock 算法适宜低延迟实现。在 4 比特 S 盒的设计中，通过在差分、线性等密码学性质最优的 16 个仿射等价类中精心选取关键路径短、面积小的 4 比特 S 盒构造。同时，线性层中大量使用了循环移位、字节置换等拉线操作，极大的降

低了加密算法每轮的时延。在此基础上，针对具体应用的时钟约束条件，可以通过以算法设计中的 2（或 4）轮作为单位，每个时钟周期内完成 2（或 4）轮加密运算，实现满足低延迟应用环境的需求。

uBlock 算法适用于 8 位微处理器实现。uBlock 算法的主要运算仅包含 4 比特 S 盒、以 4 比特/8 比特为单位的置换、以及异或等逻辑运算，非常适合 8 位微处理器实现。通过将 2 个 4 比特 S 盒并置看作一个 8 比特 S 盒，整个非线性层可以按 8 比特字节查表的方式简单实现，存储 8 比特 S 盒仅需要 256 字节 RAM。线性层中按 8 比特为单位的置换，可以通过寄存器直接操作，无需额外指令；按 4 比特为单位的置换，可以通过 swap 指令高效实现。

uBlock 算法具有良好的防护侧信道攻击的密码特性。门限实现（Threshold implementation）掩码方案是当前主流的侧信道攻击防护技术，在此以它为代表说明算法防护的低实现代价特性。一个门限防护方案就是基于门限计算和秘密共享的思想，分解输入变元和相应的函数输出，从而使得整个算法执行满足正确性、非完备性和均匀性。针对一个特定的函数，构造一个门限实现，满足输入均匀性、正确性和非完备性是容易的并且是直接的，但是其函数均匀性并不能总被满足，为满足该性质常常需要添加新的随机数。而 uBlock 算法的设计可以使得算法在一阶门限实现中无需添加新的随机数，这将大幅降低其防护代价。对于密码学性质最优的 4 比特 S 盒，文献[BNNR12]通过函数分解等技术给出了相应的门限实现方案，uBlock 算法的 S 盒属于最易门限实现的类别，可以分解为二次 F 置换和 G 置换，并且对于某些 F 置换和 G 置换可以直接找到满足函数均匀性的 3-share 的门限实现，其一阶门限实现的实现面积很低，并且在算法执行过程中无需增加新的随机数。

6. uBlock 的创新点和特色

6.1 创新点

- ✓ 整体结构面向 SSE/AVX 指令集，软件速度快；可并行，模块化，硬件实现灵活；不同参数的算法基于同一框架，便于算法扩展；采用 S 盒和分支数的理念，针对典型分析方法具有可证明安全性。
- ✓ 线性部件的设计方法通用有效，可以构造任意偶数维最优二元域扩散层，相比之前的结果具有明显优势。非线性部件的密码特性和硬件性能俱佳，关键路径短、乘法复杂度低、易于侧信道防护。
- ✓ 密钥扩展算法采用随用即生成的方式生成轮密钥，不增加存贮负担；利用向量置换构造扩展 Feistel 结构，相比广义 Feistel 结构扩散更快，相比 Feistel 结构更轻量。

6.2 算法特色

- ✓ **适应性强**

  uBlock 算法适应各种软硬件平台。PC 上可以利用 SSE、AVX2 等指令集高效实现；对于主流的 ARM 平台，可以利用 neon 指令集优化实现；非常适合 8 位微处理器实现。硬件实现方面，并行结构、模块化、简单的 S 盒和线性变换，使得 uBlock 算法的硬件实现简单而有效，既可以高速实现，也可以轻量化实现。

- ✓ **易于侧信道防护**

  uBlock 算法具有良好的防护侧信道攻击的密码特性，一阶门限实现中无需添加新的随机数，大幅降低 uBlock 算法的防护代价。uBlock 算法的 S 盒属于最易门限实现的 4 比特 S 盒，可找到满足函数均匀性的 3-share 的门限实现，一阶门限实现的实现面积很低，并且在算法执行过程中无需增加新的随机数。

- ✓ **乘法复杂度低**

  乘法复杂度（MC)的衡量指标是加密 1 比特需要的 AND 数（ANDs/bit)。

AES-128 和 AES-256 加密的乘法复杂度最低是 40 和 56；Simon-128/128 和 Simon-128/256 加密的乘法复杂度是 34 和 36；SM4 加密的乘法复杂度是 32；uBlock-128/128，uBlock-128/256 和 uBlock-256/256 加密的乘法复杂度分别是 16，24 和 24。

7. uBlock 的统计测试

根据分组密码算法检测规范设置统计检测项目和参数，uBlock 算法的统计测试结果如下。

(1) uBlock-128/128 算法

统计检测设置：

显著性水平：0.010

明文文件：高密样本

密钥文件：高密样本

工作模式：ECB

检测次数：128

各检测项目的具体检测结果如表 19 所示，通过检测的百分比阈值下限为96.362%。

表 19. uBlock-128/128 随机性统计检测结果

检测项目	通过检测的百分比(%)
重叠子序列(2)—1	99.22
重叠子序列(2)—2	100.00
单比特频数	100.00
二元推导	98.44
近似熵	99.22
矩阵秩	97.66
块内频数	100.00
块内最大游程	99.22
累加和	100.00
离散傅里叶	97.66
扑克	99.22
通用统计	96.88
线性复杂度	99.22
游程总数	98.44

游程分布	99.22
自相关	98.44
专项—分组频数	100.00
专项—分组跟随性	100.00
专项—分组明密文独立性	100.00
专项—明文雪崩	98.69
专项—密钥雪崩	98.72

(2) uBlock-128/256 算法

统计检测设置：

显著性水平：0.010

明文文件：低密样本

密钥文件：低密样本

工作模式：ECB

检测次数：128

各检测项目的具体检测结果如表 20 所示，通过检测的百分比阈值下限为 96.362%。

表 20.uBlock-128/256 随机性统计检测结果

检测项目	通过检测的百分比(%)
重叠子序列(2)—1	100.00
重叠子序列(2)—2	100.00
单比特频数	99.22
二元推导	99.22
近似熵	100.00
矩阵秩	100.00
块内频数	100.00
块内最大游程	98.44
累加和	99.22
离散傅里叶	98.44

扑克	99.22
通用统计	100.00
线性复杂度	98.44
游程总数	99.22
游程分布	99.22
自相关	99.22
专项—分组频数	98.44
专项—分组跟随性	99.22
专项—分组明密文独立性	97.66
专项—明文雪崩	98.67
专项—密钥雪崩	98.85

(3) uBlock-256/256 算法

统计检测设置：

显著性水平：0.010

明文文件：高密样本

密钥文件：低密样本

工作模式：ECB

检测次数：128

各检测项目的具体检测结果如表 21 所示，通过检测的百分比阈值下限为 96.362%。

表 21.uBlock-256/256 随机性统计检测结果

检测项目	通过检测的百分比(%)
重叠子序列(2)—1	99.22
重叠子序列(2)—2	100.00
单比特频数	98.44
二元推导	100.00
近似熵	98.44
矩阵秩	100.00

块内频数	98.44
块内最大游程	100.00
累加和	98.44
离散傅里叶	97.66
扑克	100.00
通用统计	99.22
线性复杂度	100.00
游程总数	99.22
游程分布	99.22
自相关	99.22
专项—分组频数	99.22
专项—分组跟随性	100.00
专项—分组明密文独立性	99.22
专项—明文雪崩	98.71
专项—密钥雪崩	98.92

8. uBlock 的运算实例

以下为 uBlock 算法的运算实例，用以验证密码算法实现的正确性。其中，数据采用 16 进制表示。

（1）uBlock-128/128

实例一：对一组明文用密钥加密一次

明文：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10

加密密钥：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10

密　　文：32 12 2b ed d0 23 c4 29 02 34 70 e1 15 8c 14 7d

轮密钥与每轮输出状态：

RK[0]	0123456789abcdef	fedcba9876543210
RK[1]	a1a88760f7e064ff	49c2b37e608d1f5a
RK[2]	a885d1785e4a1c57	876a080f6af41f7e
RK[3]	21b45088f5d76544	de18a5857a5c8714
RK[4]	aac1a883d0996f2c	556b748482f5140d
RK[5]	e61bb5f2f23e0669	a06c91328adfac89
RK[6]	c47f199cdedd59c5	b201eb26fef66953
RK[7]	bbed5adbd158f4e8	1e57dfcc9cd9459d
RK[8]	0637352c3e067cc2	51fe8dbedbd4b8a5
RK[9]	1fbafc2eb432761e	3e7367cc203c6250
RK[10]	d7372706f8d925a2	f47b2ae121b6fec3
RK[11]	e5c6d10462f0e004	2823976a0df5727d
RK[12]	df726151990e7efd	d2ec06400e60541f
RK[13]	264181becf8ea032	6977e21f5d9efd10
RK[14]	6867548560e60fe6	8fa4e1e3b2c06218
RK[15]	e212c92512ccacb7	5006675e866f864e
RK[16]	a5185fd1321d32eb	c2a1c25b2e1c279c

X[0]	0123456789abcdef	fedcba9876543210
X[1]	7777777777777777	7777777777777777

X[2]	f050ae409d2afbb9	40016f9e4b5a6148
X[3]	baad4aa7e0286f53	85a1af8a6d4e8980
X[4]	4b5c95363f54cf4a	1f20a4b4a7b5a9c0
X[5]	8df36718ddcda83b	98653d7076425769
X[6]	0e18e4a12ed9f9ce	a2d1679cbabbcf9c
X[7]	de247121c576223f	d27d862cb87992a7
X[8]	c88f3d3e9bcf20ff	db979d6631e69994
X[9]	bccdd8da85afc3b6	18354ff5de9b5964
X[10]	9634b15834c2db60	8a42c3dc4980466e
X[11]	304fc2fe8d75aa8b	1024e73b129f3298
X[12]	b388b82536d42325	b00a6a21e40168e1
X[13]	735943924856036a	95bc82a8793168b4
X[14]	c6f0f56e8ac0947b	d08ef42bd56762c7
X[15]	6fd0b1f487f047bb	4af6626df16076a7
密文 Y	32122bedd023c429	023470e1158c147d

实例 二：利用相同加密密钥对一组明文反复加密 1000000 次

明文：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10

加密密钥：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10

密　　文：9d 63 9e 31 06 2f fb 57 46 46 e4 28 f9 2e 08 d4

（2）uBlock-128/256

实例一：对一组明文用密钥加密一次

明文：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10

加密密钥：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10

00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

密　　文：64 ac cd 6e 34 ca c8 4d 38 4c d4 ba 7a ea dd 19

轮密钥与每轮输出状态：

RK[0]	0123456789abcdef	fedcba9876543210
RK[1]	c38120a429b54a69	e4a16e2bd396591c
RK[2]	c1dba8f52f477e36	33ec86a61e3edcfc
RK[3]	0d3843ad21b928ae	9ff31ff10cdf0d25
RK[4]	d9b6f9de2255b942	43f1dc64227136ad
RK[5]	ee8d92332ca5f258	f70251512c904359
RK[6]	014bd532ef14ffc4	123d388893ff34f0
RK[7]	9ae1f002fb3717e4	b453ebf277f68e6f
RK[8]	236b31f235692c0e	fb9c7d229b79ee9b
RK[9]	d8f3ffd8c19a2d18	25a81de235ea483f
RK[10]	0be8af5db8c58136	0b15e15145411261
RK[11]	ead60e93efb7724d	150fdc0f18d87103
RK[12]	82fa62ffb56dfcd4	ecf30fb71fa52355
RK[13]	7a2ff89920da4d35	93a49548b2d1ab21
RK[14]	0863541222a81aa0	36da8afec9173056
RK[15]	594ad784870069b1	697949815563987f
RK[16]	d4005b8559e76b86	0ec671c760b002b0
RK[17]	8b35a59f96d4fe0f	c74540d9b76a78b9
RK[18]	272e14f270d31b7a	b5aae60a36dcc0e8
RK[19]	553ab6324201c21e	b9638f6d53d84941
RK[20]	9ac70d41f9ca4743	7231a3f92ed743f9
RK[21]	4e651f7f02189773	497bb493f7de2cc3
RK[22]	511fdd53fa7f8900	13a68a41c5e1ffbf
RK[23]	1e8c3f6f3344f1c2	a5c6b25a5165d200

RK[24]	6270fb691e63aafd	dfcf87f97f644b5d
X[0]	0123456789abcdef	fedcba9876543210
X[1]	7777777777777777	7777777777777777
X[2]	ebb5445e6a968fbf	d6e33bd6aa208395
X[3]	5f20a01cb9b368f0	1011a92cb7d73d30
X[4]	47ee34f851566027	5755eb613cd6fc88
X[5]	11c1374400036304	2044d270dea4f36c
X[6]	6fbaeea5d4767cfe	94cc0427a910e7f7
X[7]	d919d44e94fc0cb4	8a4f301fb1b96658
X[8]	126ace021eaebd43	220f401e9de3f336
X[9]	1be94af4aceccdba	2a09326558e215c6
X[10]	3ae764c1e08ad81d	e71455b0cc392997
X[11]	ee60b05b7aa3935a	39f202254e0e1449
X[12]	4547b88d265fb149	620e4fbb7c2298c5
X[13]	e9655da79cc3bf30	cffb915912af60e2
X[14]	f5bccc6a11dd17bc	74f6752ac5aeb2a1
X[15]	a558c1c108f393e6	fa266510b009f51d
X[16]	e22c03565474a2a7	2c4c14beea5cf545
X[17]	df4f67c9114f79d2	82aed10f343a9685
X[18]	e58cda633cdb92fe	2a6edcfa5b7b70ce
X[19]	39bc631c1a693085	2bc887f91594eabf
X[20]	979324911f9e304a	c4ae83d79aaec0e2
X[21]	5342df25eb8b1c96	bd5ce43677997e71
X[22]	82c53b6c0c3680c9	42e728c3a096d65f
X[23]	d4d08f47ca2ba1fa	cd605c958e3ceb26
密文 Y	64accd6e34cac84d	384cd4ba7aeadd19

实例 二：利用相同加密密钥对一组明文反复加密 1000000 次

明文：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10

加密密钥：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10

00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

密　　文：9f ca 87 e0 fb ae bc 91 05 e7 29 d3 55 39 67 ff

（3）uBlock-256/256

实例一：对一组明文用密钥加密一次

明 文：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

加密密钥：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

密 文：d8 e9 35 1c 5f 4d 27 ea 84 21 35 ca 16 40 ad 4b
0c e1 19 bc 25 c0 3e 7c 32 9e a8 fe 93 e7 bd fe

轮密钥与每轮输出状态

RK[0]	0123456789abcdeffedcba9876543210	000102030405060708090a0b0c0d0e0f
RK[1]	c38120a446b5e0cae4a16e2bd396591c	76543210fedcba98a5f0278d1e4c9b36
RK[2]	c1eba84a3e4e60d933ec86a6e63e5473	d396591ce4a16e2bb0ac84403c2e651a
RK[3]	0d88b8b7a7bc4c404cc32ca65ad55966	e63e547333ec86a6489cea301da6eeb4
RK[4]	20d6fea208870650ef064bfc9610a5c8	5ad559664cc32ca6b80087acd4b47c8b
RK[5]	e4e1e29d3694d477a420fd1f29b8f950	9610a5c8ef064bfc8e02d20605f0876a
RK[6]	a58af90a572590b2cf5103a67db05e47	29b8f950a420fd1f927eed3447ed6419
RK[7]	a16f03e6027e77087b2e42bafa4f6265	7db05e47cf5103a6292a8a505bf975a0
RK[8]	0899e885882991f4fea74e9abeb6f1f7	fa4f62657b2e42ba738a660710072efe
RK[9]	be9c31368e00bc552cd6b4d9cf79a1e9	beb6f1f7fea74e9a284095818fe98998
RK[10]	d7e67a76d461699695a1e770c475d109	cf79a1e92cd6b4d9015b968ce53be0c3
RK[11]	cc4be62fa25b0001b9b1048dff491e23	c475d10995a1e7706a6de6d979764167
RK[12]	fb8836df92867119b7ecc99f2824495e	ff491e23b9b1048d561c4fa0c0e02bb2
RK[13]	5ee1675cdd79997e1554ce0f9e1b9aaf	2824495eb7ecc99f869f8f91b137268d
RK[14]	5dda43759f32f7d599ba98ec9b9f9980	9e1b9aaf1554ce0f77e5ecd9e769d915
RK[15]	e29ff18626e1971762f8fde9f63ecaee	9b9f998099ba98ec3355d597dd4ff2a7
RK[16]	2c99448c0de44cb3ef3a6268199c2b56	f63ecaee62f8fde9e17e962721f961f8
RK[17]	2b257fe86443c606c065d38bdc9ea53b	199c2b56ef3a6268e4329c0ccb44d498
RK[18]	e8f04b7b88badeca7f2de6697b8e42ad	dc9ea53bc065d38b4f622866b07c435e
RK[19]	88c982cf30d622f3aa5dac1f5d35bf9f	7b8e42ad7f2de669bbaefb8e8c4d8a07
RK[20]	830044348ad49b8912c4a693fbe4f12a	5d35bf9faa5dac1fd238cf328f82069c
RK[21]	381d7d8565e3b76307c01d75ab3a7a8a	fbe4f12a12c4a693d498048b3849a403
RK[22]	aa235de0ef420aaca1ffe3bc36f98de9	ab3a7a8a07c01d75ed331567867b53d8
RK[23]	b2fe983d166744b781e22c96f4a6a583	36f98de9a1ffe3bc4dca20eaaa50f23e
RK[24]	be7ee15b37143b9095b65adc98a4b922	f4a6a58381e22c96687bfd142b9467e3

X[0]	0123456789abcdeffedcba9876543210	000102030405060708090a0b0c0d0e0f
X[1]	77777777777777777777777777777777	77777777777777777777777777777777

X[2]	9829121062f599ea89176e87ef338dd1	8a38aaf9ae316d5ee40157177c4cb691
X[3]	688e087be6344cfb846ab56a1fa2846d	b03712a50029f54a46604f8f044d947f
X[4]	8a0fdcf6a29107938c38e760f544fc64	2f9fc1d95535d909d6095e766142e1a6
X[5]	99f6c70aae84782f8abdf6a81af9eba0	e807771ca6d80ba47f73a3b35cbeb113
X[6]	df179e654544f7472d2fd9449d426b0e	dc5c14a9ddc238a6f13afce4acde5b7b
X[7]	1a05810894319dc3ae63198dd822484f	c80451543b53ccdfdddac089bd4ba285
X[8]	31a339370b84a72ac0e19326c45fb996	e462f743ecc5cd74efbecd2d1cbc2067
X[9]	fa548abb8c8e6dd08286cbd7641c4ca6	4f939161e542dee33324afafa7cd7507
X[10]	885631c97868e9de702017685af25ba1	9bb2880bb97fa3eeaf01ca169eded34e
X[11]	bf2287a752de752b4a4dd308ab3c97b6	1f03b279859b82b56da69824c78c509d
X[12]	e744e5b851ff50ca9074fe2eb955795f	b3220272b998e5503b76650c61713846
X[13]	a1650de67dc2060427a4f1dec00e295f	e46f2bf3759d6b06503f6ade78fd47d7
X[14]	cfea45d03bb58fa5dd3db7fc11597737	00dfba6b1205ead7cd2da4cd6201b5b2
X[15]	26085b76d52545267dc9feef18600fc7	77ba94bd6f7fbebf5382cec221dd8d23
X[16]	0c651bf049940023fc970b21ccbacc1c	cf2b459239a51caa31eaac4d53b95c3d
X[17]	e47b80c02585163904131bdedc778627	42d72488d2ab4791fe9e126afc0ca38a
X[18]	5b482301c3b45af6a9d4d959ab03635c	0bc52c352b199d82da23f154f0c5c42b
X[19]	108e7aae4fb472ceb393a22bc64256b1	d34edb3452b803bb61985b137f52a660
X[20]	206f7ad78f5b4709fb139c4e90861f88	3f4172449abeb24cfa9f03f45a472711
X[21]	86a14ee4a7620351392d732bf282a63f	8bdf41d52e4606ca32fb07fb4fdb80fa
X[22]	0a04483f2004ebf0e7e69e52ca60b9d7	57aa8c3360dfc7955b3db97abb05ba53
X[23]	9d23c633640621824dac86b48001b544	2769b2100af299be1f647dde4acbf141
密文 Y	d8e9351c5f4d27ea842135ca1640ad4b	0ce119bc25c03e7c329ea8fe93e7bdfe

---

实例二：利用相同加密密钥对一组明文反复加密1000000 次

明 文：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

加密密钥：01 23 45 67 89 ab cd effe dc ba 98 76 54 32 10
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

密 文：20 e1 48 e6 eb f9 d9 9b 89 4b 2f 7e 5c cd e2 3f
c5 8c 95 6c 78 47 fa d8 2e 08 24 41 35 76 c8 9f

致谢：算法硬件实现性能评估得到清华大学苏冠通博士的大力支持，在此表示感谢。

附录参考文献

[AIK01] Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita,T.: Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms-Designand Analysis. SAC2000. LNCS 2012, pp.39 - 56, Springer Heidelberg (2001).

[AMB13] AnneCanteaut, MarıaNaya-Plasencia, Bastien Vayssiere: Sieve-in-the-Middle: Improved MITM Attacks. CRYPTO 2013. LNCS 8042,pp.222 - 240, Springer Heidelberg.

[BBS04] Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31Rounds Using Impossible Differentials. EUROCRYPT2004. LNCS 3027, pp.12 - 23, Springer Heidelberg.

[BKR11] Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of theFull AES. ASIACRYPT 2011. LNCS7073,pp.344 - 371, Springer Heidelberg.

[BS93] Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard.Springer, Berlin (1993)

[DKR97] Daemen J, Knudsen L, Rijmen V. The block cipher Square. FSE 1996.LNCS 1267,pp.149-65, Springer Heidelberg(1997).

[BNNR12] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, Georg Stütz. Threshold Implementations of All 3×3 and 4×4 S-Boxes. CHES 2012. LNCS 7428 ,pp.76-91, Springer Heidelberg.

[LW17] Li L, Wu W.: Improved meet-in-the-middle attacks on reduced-round Kalyna-128/256 and Kalyna-256/512. Designs Codes & Cryptography, 2017, 86(4):1-21.

[JJIY14] Jian Guo, Jérémy Jean, Ivica Nikolic, Yu Sasaki: Meet-in-the-Middle Attacks on Generic Feistel Constructions. ASIACRYPT 2014. LNCS 8873, pp.458-477, Springer Heidelberg.

[GWG15] Guo Z, Wu W, Gao S. Constructing lightweight optimal diffusion primitives with Feistel structure. SAC 2015. LNCS 9566, pp.352-372, Springer, Heidelberg (2016)

[P07] Poettering, B.: RijndaelFurious AES-128 Implementation for AVR Devices (2007),http://point-at-infinity.org/avraes/

[KW02] Knudsen, L., Wagner, D.: Integral Cryptanalysis.FSE 2002. LNCS 2365,

pp.112－127. Springer, Heidelberg (2002)

[M94] Matsui, M.: Linear Cryptanalysis Method for DES Cipher.EUROCRYPT 1993. LNCS 765, pp.386－397. Springer, Heidelberg (1994)

[SSA07] Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit BlockcipherCLEFIA (Extended Abstract). FSE 2007. LNCS 4593, pp.181－195. Springer, Heidelberg (2007)

[WM12] Wu S , Wang M . Automatic Search of Truncated Impossible Differentials for Word-Oriented Block Ciphers[J]. 2012.

[WZ11] Wenling Wu and Lei Zhang. LBlock: A lightweight block cipher. ACNS 2011, LNCS 6715, pp.327－344. Springer,Heidelberg(2011).

[XZB16] Xiang Z, Zhang W, Bao Z, et al. Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. ASIACRYPT 2016. LNCS 10031, pp.648-678. Springer, Heidelberg (2016).

[YT15] Todo Y. Structural Evaluation by Generalized Integral Property. EUROCRYPT 2015, pp.287－314.Springer, Heidelberg (2015).

[YT16] Todo Y, Morii M. Bit-Based Division Property and Application to Simon Family. Revised Selected Papers of the International Conference on Fast Software Encryption. 2016.

[ZW15] Zhang H, Wu W. Structural Evaluation for Generalized Feistel Structures and Applications to LBlock and TWINE. INDOCRYPT 2015. pp.218-213. Springer, Heidelberg (2015)