成绩	

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

密码工程

分组密码软件实现

题　　目： 分组密码 uBlock 调研报告

学　　院： 网络空间安全学院

专业名称： 网络空间安全类

学　　号： 1120232895

姓　　名： 李俊霏

任课教师： 丁瑶玲 王安

评 阅 人：

目录

分组密码 uBlock 调研报告

摘要 本文研究了分组密码算法 uBlock 以及其在 ECB 和 CBC 两种工作模式下的应用。首先，对 uBlock 算法的设计背景、技术特点、密钥扩展机制和加解密流程进行了分析。其次，在实现方面，本文结合 C++编程和 SIMD 指令集，设计了 uBlock 的核心功能模块，并优化了吞吐量表现，实验验证了其加密正确性和高效性。测试结果显示 uBlock 相比传统 AES 算法在吞吐量和稳定性上表现更优。此外，经过图像加密实验，基于 BMP 图像文件分别实现了 ECB 和 CBC 两种加密模式。研究结果表明，uBlock 在保持较高安全性的同时，具有良好的实现效率和工程实用性，可广泛应用于信息安全领域。

关键词 分组密码；uBlock；ECB 模式；CBC 模式；吞吐量；性能测试；图像加密

1 uBlock 算法调研

1.1 算法背景

为贯彻落实习近平网络强国战略思想和党的十九大精神，繁荣我国密码理论和应用研究，推动密码算法设计和实现技术进步，促进密码人才成长，中国密码学会决定举办全国密码算法设计竞赛。2019 年 10 月 25 日，中国密码学会发布了全国密码算法设计竞赛的通知。

2019 年，由中国科学院软件研究所的吴文玲、张蕾、郑雅菲、李灵琛共同设计而来的 uBlock 算法在分组密码赛道中斩获一等奖，成为国内轻量级密码算法研究的代表性成果。

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，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。

1.2 uBlock 使用的符号

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
PK^i	n 比特轮密钥
PL_n，PR_n，PL^{-1}，PR_n^{-1}	$\frac{n}{16}$ 个字节的向量置换
s	4 比特 S 盒
S_n，S_n^{-1}	$\frac{n}{8}$ 个s盒的并置
S_k	$\frac{k}{16}$ 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
$\lll b$	循环左移 b 比特
$\lll_{32} b$	分款 32 比特循环左移 b 比特
$\|\|$	表示比特串的连接

表 1 uBlock 使用的符号

1.3 技术文档

见附件《uBlock 算法设计文档》

2 uBlock 系统设计

2.1 总体框架

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。

2.1.1 uBlock 整体结构

uBlock 算法的整体结构称为 PX 结构，如图 1 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 4m+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$P_j(j=1,\ldots,m)$、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。$P_j(j=1,\ldots,m)$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$b_j(j=1,\ldots,m)$。

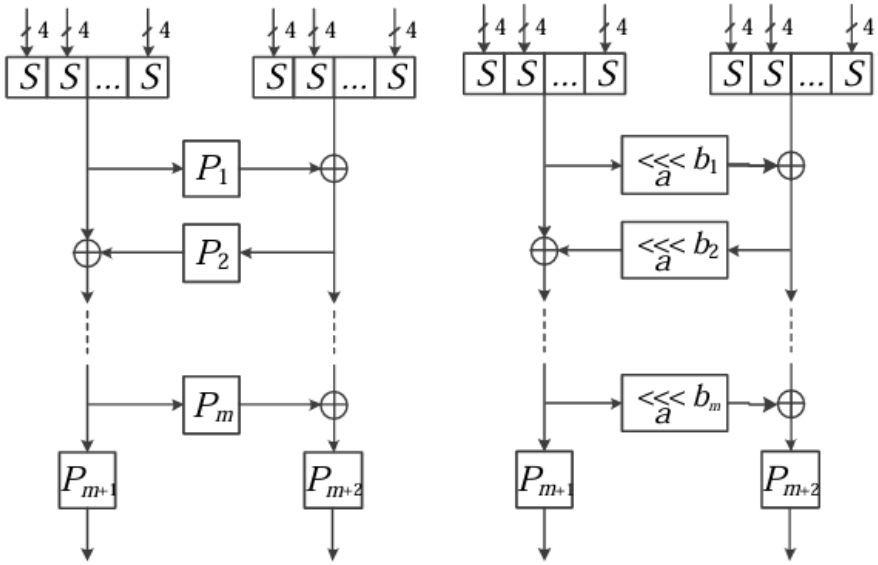


图 1 PX 整体结构

2.1.2 uBlock 加密

加密算法由 r 轮迭代变换组成，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。

下图是 uBlock 的加密伪代码与流程图示：

$uBlock.Enc(X, RK^0, RK^1, ..., RK^r)$
$X_0 \| X_1 \leftarrow X$
for $i=0$ *to* r-1 *do*
$RK_0^i \| RK_1^i \leftarrow RK^i$
$X_0 \leftarrow S_n(X_0 \oplus RK_0^i)$
$X_1 \leftarrow S_n(X_1 \oplus RK_1^i)$
$X_1 \leftarrow X_1 \oplus X_0$
$X_0 \leftarrow X_0 \oplus (X_1 <<<_{32} 4)$
$X_1 \leftarrow X_1 \oplus (X_0 <<<_{32} 8)$
$X_0 \leftarrow X_0 \oplus (X_1 <<<_{32} 8)$
$X_1 \leftarrow X_1 \oplus (X_0 <<<_{32} 20)$
$X_0 \leftarrow X_0 \oplus X_1$
$X_0 \leftarrow PL_n(X_0)$
$X_1 \leftarrow PR_n(X_1)$
$Y \leftarrow RK^r \oplus (X_0 \| X_1)$

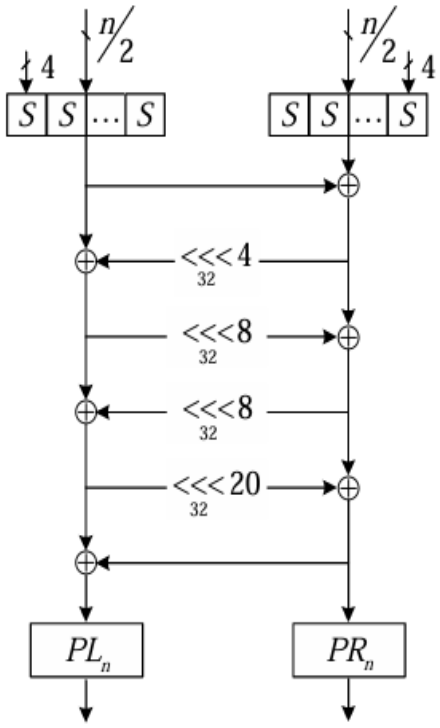


图 2 uBlock 的加密伪代码(左)与流程图(右)

从图中，我们可以看出我们一共要加密 r 轮，r 的值取决于明文与密钥的比特值。先讲 n 比特的明文拆分成两个 n/2 比特的子密文 X_0 和 X_1，在每轮中：

1. 我们需要把轮密钥拆分成两个轮子密钥 RK_0^i 和 RK_1^i
2.我们先将子密文和轮子密钥 RK_0^i 做模 2 的加法，然后过一个 S_n，得到新的 X_0

S_n 由 8/n 个相同的 4 比特盒并置而成，定义如下：

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

4 比特盒如表 2 所示。

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

表 2 4 比特盒

3. 我们同样对 X_1 和 RK_1^i 异或且过 S 盒，得到新的 X_1
4. X_0 和 X_1 异或后得到新的 X_1。
5. 将 32 位分块 X_1 循环左移 4 比特后与 X_0 异或得到新的 X_0
6. 将 32 位分块 X_0 循环左移 8 比特后与 X_1 异或得到新的 X_1
7. 将 32 位分块 X_1 循环左移 8 比特后与 X_0 异或得到新的 X_0
8. 将 32 位分块 X_0 循环左移 20 比特后与 X_1 异或得到新的 X_1

9. X_0 和 X_1 异或得到 X_0

10.对 X_0 和 X_1 进行 n/16 字节的向量置换

对于 $PL_n(X_0)$和 $PR_n(X_1)$的定义如下：

$PL_n(X_0)$和 $PR_n(X_1)$都是 n/16 字节的向量置换具体见表 3，对于 PL_{128} 的表达式为：

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

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}

表 3 PL_n 和 PR_n

最终的密文 Y 即为第 r 轮密钥 RK^r 与最终得到的 X_0 和 X_1 的并置进行异或。

2.1.3 uBlock 解密

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，$RK^{r-1}\cdots RK^0$，输出 n 比特明文 X。

下图是 uBlock 解密伪代码：

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

图 3 uBlock 的解密伪代码

中的 S_n^{-1}、PL_n^{-1}、PR_n^{-1} 分别是 S_n、PL_n 和 PR_n 的逆，具体见表 4 和表 5。

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 S_n^{-1} 表

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}

表 5 PL_n^{-1} 和 PR_n^{-1}

解密过程和加密过程一致，不再赘述

2.1.4 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 $1, 2, \ldots, r$，更新寄存器，并取寄存器的左 n 比特作为轮密钥 RK^i。

寄存器更新方式和密钥状态更新函数如下：

$$K_0 \| K_1 \| K_2 \| K_3 \leftarrow K$$
$$K_0 \| K_1 \leftarrow PK_t(K_0 \| K_1)$$
$$K_2 \leftarrow K_2 \oplus S_k(K_0 \oplus RC_i)$$
$$K_3 \leftarrow K_3 \oplus T_k(K_1)$$
$$K \leftarrow K_2 \| K_3 \| K_1 \| K_0$$

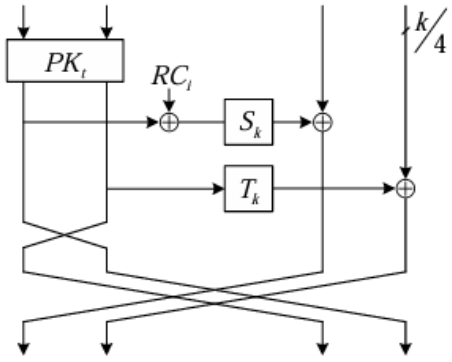


图 3 寄存器更新方式(左)和密钥状态更新函数(右)

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⊗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是 32 个半字节的向量置换，具体见下表。

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}

表 6 PK_1、PK_2和 PK_3向量置换

32 比特常数 RC_i由 8 级 LFSR 生成，初始条件为$c_0 = c_3 = c_6 = c_9 = 0$，$c_1 = c_2 = c_4 = c_5 = 1$；对于 i≥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。

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

图 4 常数 RC_i

2.2 数据结构设计

我们以 uBlock_Windows_128_128.cpp 为例，设计我们 uBlock 的数据结构

2.2.1 轮密钥数组 Subkey[17][32]

轮密钥数组的建立是非常有必要的。

从安全性的角度分析，我们使用轮密钥加密可以分散主密钥的信息。如果我们在加密的过程中使用同一个密钥，攻击者很容易能找出密钥的规律，我们使用轮密钥，即使攻击者获取了一部分的轮密钥也很难还原成主密钥。从一定程度上增加了抵抗密码算法的非线性特征，让差分密码等破解的难度显著提高。

从加密效率来说，我们在现代电脑中运行 uBlock 这种分组密码算法，我们可以充分利用并行计算的能力，我们可以通过 SSE 等系统调用快速运算。同时，如果我们在算法的每一轮重新计算轮密钥，那么我们每一轮计算的负担会显著增加，浪费加密之间的准备时间。

2.2.2 轮常数 RC[16][16]

轮常数 RC 的设置也非常有必要。

轮常数可以确保每一轮密钥的独立性，如果我们的算法中没有设计轮常数，那么密钥扩展算法有可能生成重复的轮密钥，另一个角度说，轮常数为轮密钥+提供了额外的随机性，增加密钥的多样性。

当然，增加了密钥的随机性，就增加了攻击的难度，很难通过轮密钥之间的关系来推出主密钥。

另外，如果我们没有轮常数，我们需要其他的方式来增加密钥的随机性，可能是算一个 hash，所以我们设计的轮常数为密钥扩展算法提供一个稳定的参考点，在每一轮的密钥扩展中，轮常数作为一个固定的值，让密钥扩展的过程变得简洁和易于实现。

2.2.3 SIMD 向量寄存器(SSE 指令集)

SIMD（Single Instruction, Multiple Data，单指令多数据）向量寄存器及其相关的指令集（如 Intel 的 SSE 指令集）在算法中设计是非常有必要的。

从数据处理速度来说，SSE 指令集充分利用了现代计算机 CPU 的并行能力，从 1999 年的 SSE 版本到 2008 年发售的 SSE4.2 版本，计算机并行计算能力在增强。

指令集	Intel 支持起始	AMD 支持起始
SSE	Pentium III (1999)	Athlon XP (2001)
SSE2	Pentium 4 (2001)	Athlon 64 (2003)
SSE3	Prescott P4 (2004)	Athlon 64 (部分指令)
SSSE3	Core 2 Duo (2006)	Bulldozer (2011)
SSE4.1	Penryn Core 2 (2007)	K10 (2007)
SSE4.2	Nehalem Core i7 (2008)	Bulldozer (2011)

表 7 SSE 指令集发展过程我们可以在个人笔记本上看出 CPU 支持这个指令

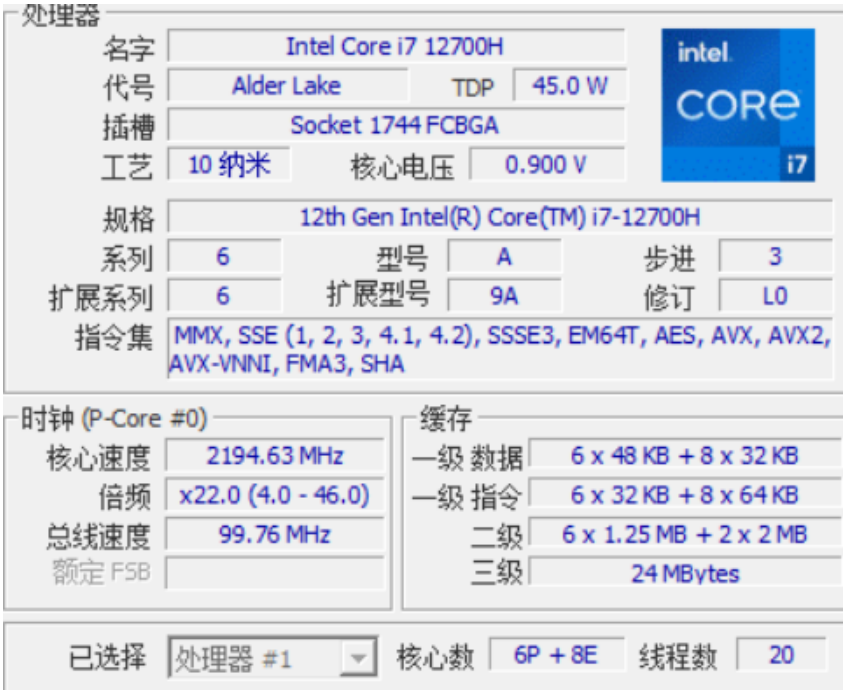


图 5 个人笔记本支持指令集展示

SSE4.2 增强了 CPU 对 SIMD（单指令多数据）并行计算的支持，主要优化了字符串处理、文本检索、数据校验等场景。

在我们算法中，我们主要运用_mm_shuffle_epi8，主要用于 S 盒 16 个并行查找。

运用_mm_xor_si128，在轮密钥混合中，并行完成 128 位状态矩阵与轮密钥的异或

运用_mm_shuffle_epi8 + 自定义控制掩码，实现行列置换等等。

2.2.4 状态矩阵

在分组密码中，明文和密文通常是被分成若干固定大小的块中，状态矩阵就提供了一种方便的模式去存储这个数据块。

在明文与密文在矩阵中存储，有效简化了加密中的矩阵运算、执行逆变化，以及便于 SSE 进行并行处理。

2.2.5 置换控制向量

置换控制向量用于定义数据元素（如字节、位或整数）在置换操作中的目标位置。它是一个索引向量，指定了每个数据元素在置换后的新位置。例如，在一个简单的字节置换中，置换控制向量可以定义为 [3, 0, 1, 2]，这意味着第一个字节将被移动到第四个位置，第二个字节移动到第一个位置，依此类推。

置换控制向量可以与 SIMD（单指令多数据）指令集结合使用，实现高效的向量化处理。例如，使用 SSE 指令集中的 _mm_shuffle_epi8 指令，可以根据置换控制向量快速地重新排列数据，从而提高数据处理的效率。

3 算法实现

3.1 流程图设计

算法设计源文件附后(uBlock_128.cpp)，源代码由 uBlock 算法-2-算法实现源代码-uBlock_Windows_128_128.cpp 改写而来。

算法设计主要由三大部分设计，分别为密钥生成子算法、加密子算法和解密子算法。

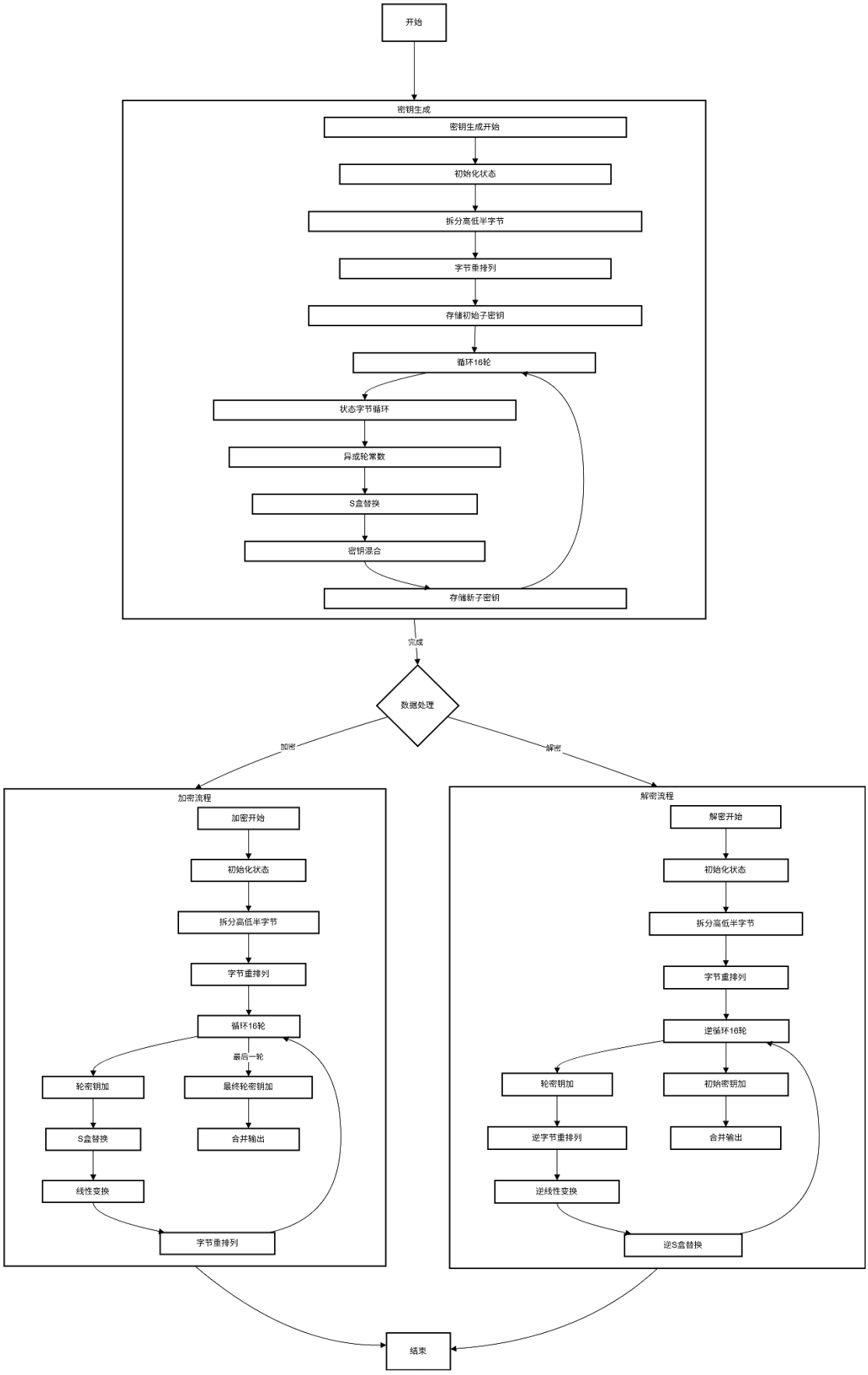

开始
密钥生成
密钥生成开始
初始化状态
拆分高低半字节
字节重排列
存储初始子密钥
循环16轮
状态字节循环
异或轮常数
S盒替换
密钥混合
存储新子密钥
完成
数据处理
加密
解密
加密流程
加密开始
初始化状态
拆分高低半字节
字节重排列
循环16轮
最后一轮
轮密钥加
最终轮密钥加
S盒替换
合并输出
线性变换
字节重排列
解密流程
解密开始
初始化状态
拆分高低半字节
字节重排列
逆循环16轮
轮密钥加
初始密钥加
逆字节重排列
合并输出
逆线性变换
逆S盒替换
结束

图 6 uBlock 算法设计流程图

3.2 核心功能实现

3.2.1 核心函数说明

代码的核心函数结构如下：

```
void uBlock_128_KeySchedule(unsigned char *key)

void uBlock_128_Encrypt(unsigned char *plain, unsigned char *cipher, int round)

void uBlock_128_Decrypt(unsigned char *cipher, unsigned char *plain, int round)

int Crypt_Enc_Block(unsigned char *input, int in_len, unsigned char *output, int *o
ut_len, unsigned char *key, int keylen)

int Crypt_Dec_Block(unsigned char *input,int in_len, unsigned char *output,int *out
_len, unsigned char *key, int keylen)

int Crypt_Enc_Block_Round(unsigned char *input, int in_len, unsigned char *output,
int *out_len, unsigned char *key, int keylen, int cryptround)

int Key_Schedule(unsigned char *seedkey, int keylen, int direction, unsigned char *
subkey)
```

3.2.2 具体设计

uBlock_128_KeySchedule 根据输入的 128 位主密钥生成 17 轮子密钥，具体流程为：

初始化常量（SK,c1-c8 等），加载主密钥并分成两个状态 state1 和 state2，通过置换和异或操作生成初始子密钥，最后使用轮常数和 S 盒进行 16 轮迭代，生成后续的轮子密钥。

uBlock_128_Encrypt 对 128 位明文进行加密并输出密文，具体流程为：

初始化置换表（L1,L2,c1-c8 等），加载明文并分成两个状态 state1 和 state2，并进行轮加密，每轮包含：轮密钥加，S 盒替换，扩散，置换，最后合并状态并输出密文。

uBlock_128_Decrypt 对 128 位密文进行解密并输出明文，具体流程类似于加密流程，但操作顺序相反，同时使用逆 S 盒 S_Inv 和逆线性变换，轮密钥从后往前使用。

Crypt_Enc_Block 调用 uBlock_128_Decrypt 对任意长度的输入数据进行分块解密；Crypt_Enc_Block_Round 调用 uBlock_128_Encrypt 对任意长度的输入数据进行分块加密，且轮数可选。

Key_Schedule 调用 uBlock_128_KeySchedule 生成子密钥，且根据 direction（0 为加密，1 为解密）将子密钥按顺序或逆序导出到 subkey 中。

具体代码见附件 uBlock_128.cpp。

4 实验验证

4.1 正确性验证

在 uBlock 设计文档中，有关于明文密钥和密文的测试用例，我们编写 uBlock_correction.cpp，验证其正确性。

程序流程图如下：

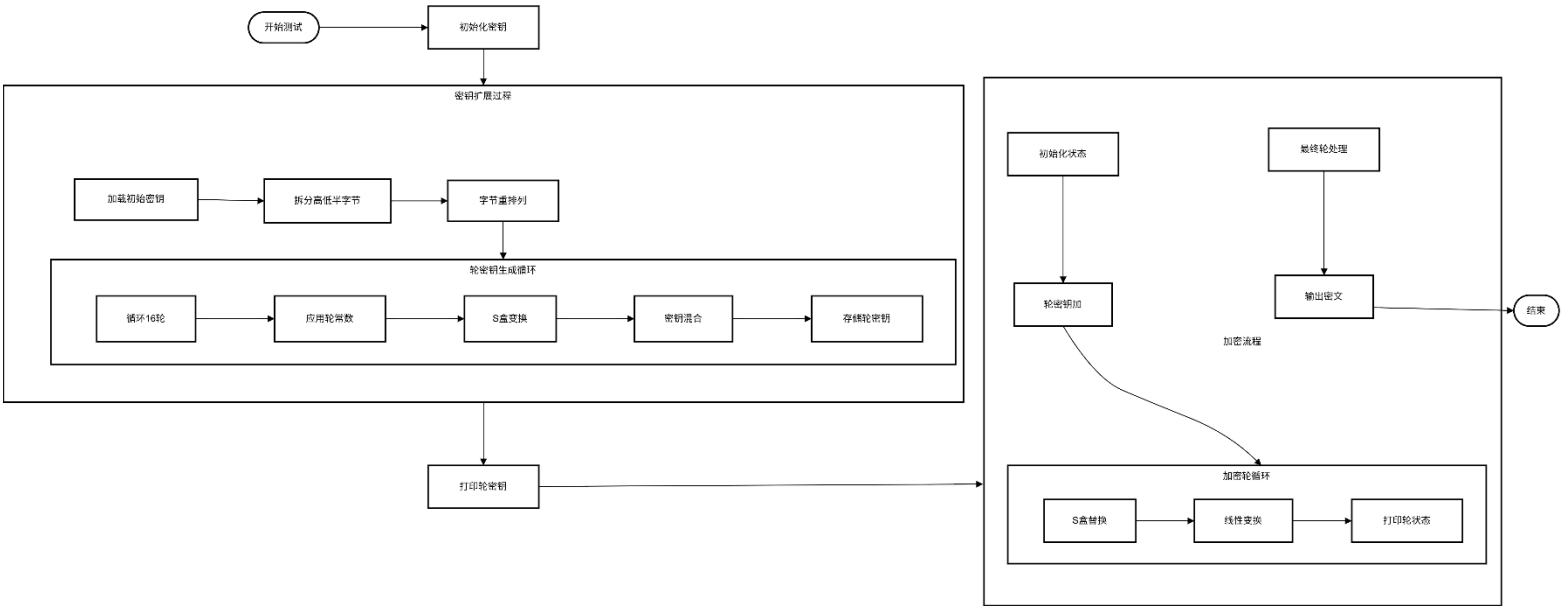


图 7 验证正确性程序流程图

主要增加打印轮密钥和在每轮加密中打印轮状态。

将明文和密钥设置为：

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

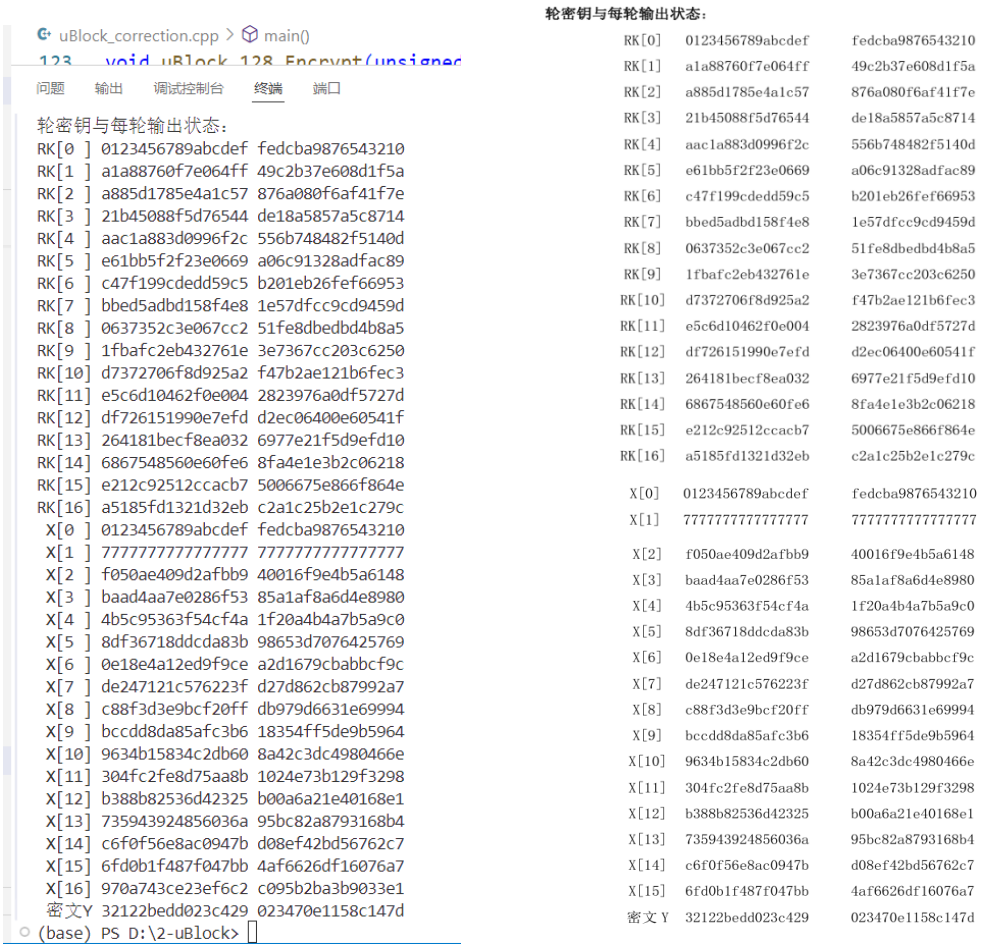


图 8 设计正确性程序输出（左）与官方文档标准输出（右）

我们可以得到以上结果，对比我们设计设计的验证正确性程序输出与官文文档中标准输出做对比，发现一致。

4.2 性能评估

4.2.1 性能测试

我们采用加密固定大小的数据（160MB）来测量加密的吞吐量（ECB 模式），为了更好的了解吞吐量大小，我们引入 openssl 中 aes 加密的部分一并测试。

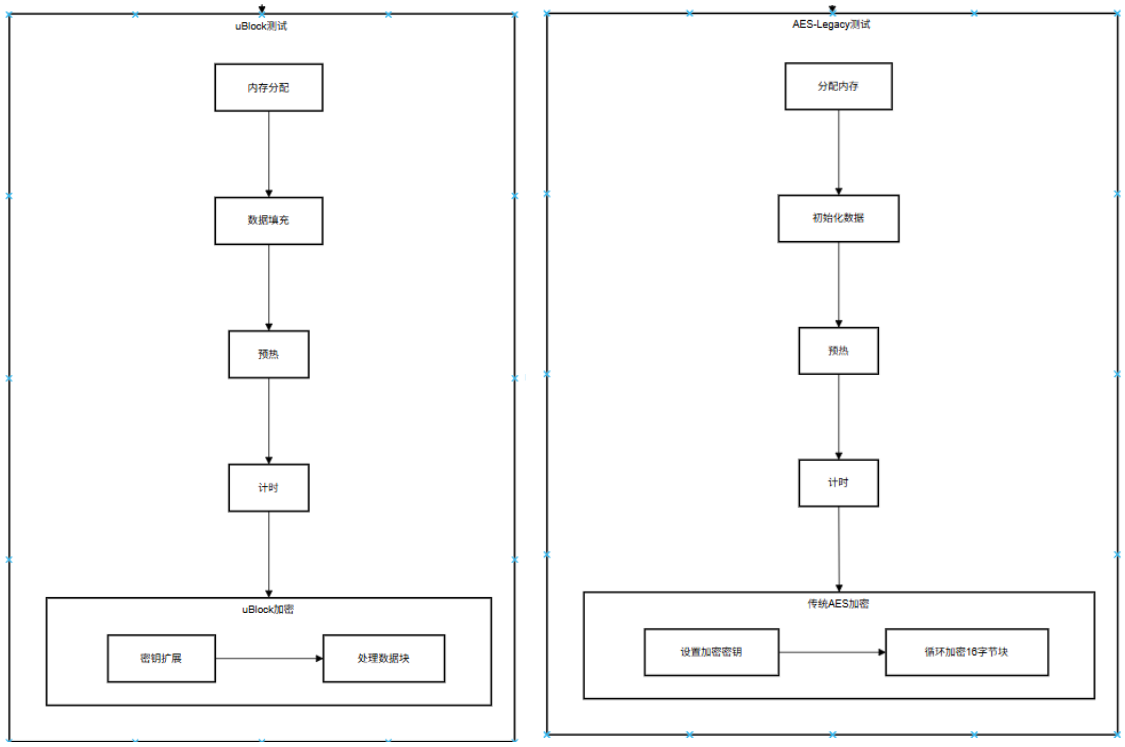
具体测试加密方案如下：

AES-EVP (OpenSSL)：使用 OpenSSL 的 EVP 接口，OpenSSL 推荐的现代加密方式。

AES-Legacy：使用 OpenSSL 的传统 AES 加密函数（AES_set_encrypt_key 和 AES_encrypt），这些函数在 OpenSSL 3.0 中被标记为弃用。

uBlock-128：使用自定义的 uBlock 加密库实现。

我们设计 uBlock_time.cpp 用于测试 uBlock 的吞吐量，下图为程序流程图



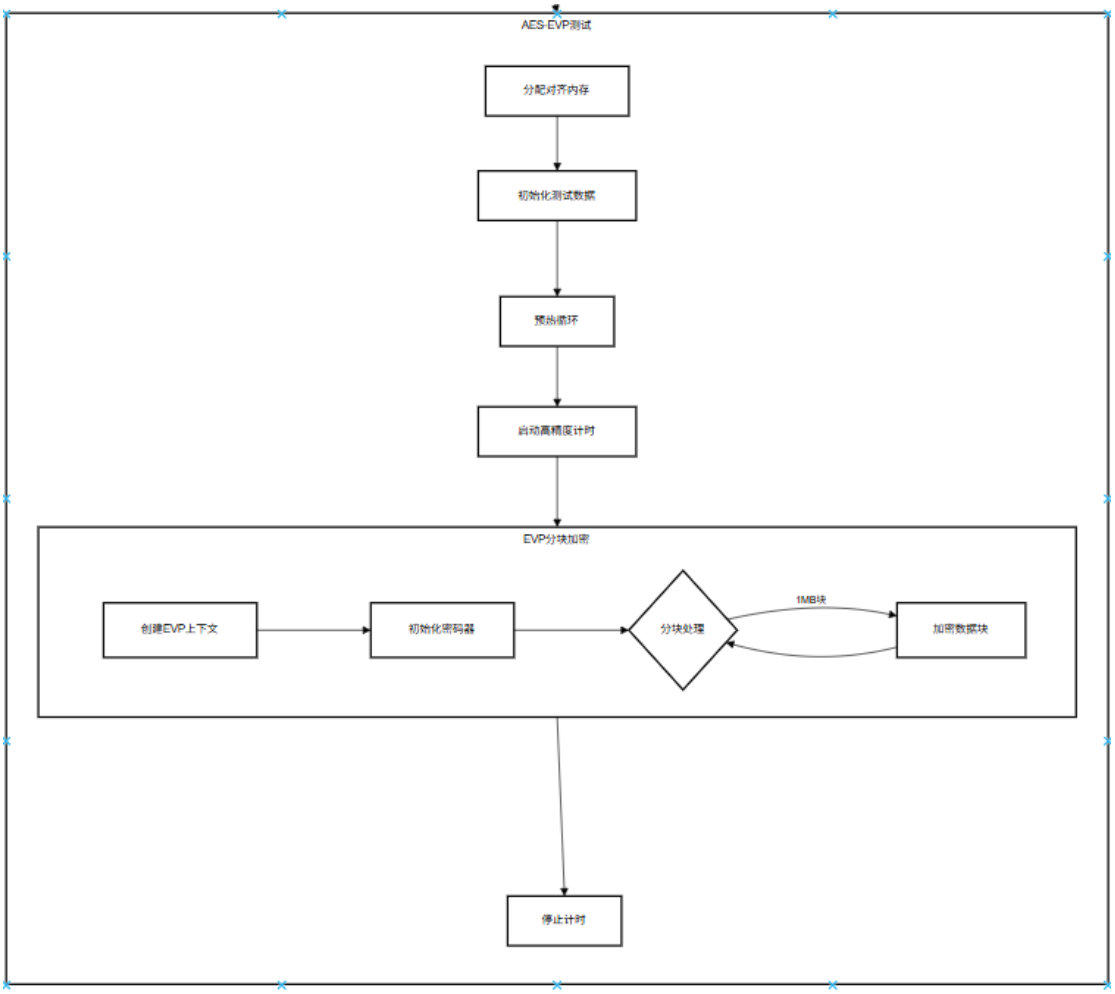


图 9 吞吐量程序流程图

为了更好的对比吞吐量，我做出以下优化：

```
    // 1.预热（减少冷启动误差）
    int out_len;
    for (int i = 0; i < WARMUP_ROUNDS; i++) {
        encrypt_func(plaintext, BLOCK_SIZE, ciphertext, &out_len, key,
key_len);
    }
    // 2.高精度计时
    LARGE_INTEGER start, end, freq;
    QueryPerformanceFrequency(&freq);
    QueryPerformanceCounter(&start);
    QueryPerformanceCounter(&end);
    // 3.执行加密（一次性处理全部数据）
    encrypt_func(plaintext, TOTAL_DATA_SIZE, ciphertext, &out_len, key,
 key_len);
    // 4.统一测试密钥
    unsigned char key[16] = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD,
0xEF,
                             0xFE, 0xDC, 0xBA, 0x98, 0x76, 0x54, 0x32, 0
x10};
```

我测试的环境为

Visual Studio Code 1.99.3;

OpenSSL 3.0.15 3 Sep 2024 (Library: OpenSSL 3.0.15 3 Sep 2024);

g++ (x86_64-posix-seh-rev2, Built by MinGW-Builds project) 14.2.0;

处理器 12th Gen Intel(R) Core(TM) i7-12700H，2300 Mhz，14 个内核，20 个逻辑处理器

系统型号 ASUS TUF Gaming F15 FX507ZC4_FX507ZC4

操作系统 Microsoft Windows 11 家庭中文版

磁盘驱动器 NVMe KINGBANK KP230 Pro *2

优化项：在 task.json 文件中，增添 argc："-O3"，" -march=native"

在测试中，增添优化项，将 uBlock128 解密吞吐量增加 16 倍左右，即从 13.6MB/s 左右提升至 231.66MB/s 左右

输出如下：

```
(base) PS D:\2-uBlock>  & 'c:\Users\23979\.vscoc
icrosoft-MIEngine-In-xgpf3b00.vmq' '--stdout=Mic
ngine-Pid-asvxw5un.ett' '--dbgExe=C:\Program Fil
[AES-EVP (OpenSSL) ] 吞吐量: 3193.77 MB/s
[AES-Legacy        ] 吞吐量: 145.73 MB/s
[uBlock-128        ] 吞吐量: 237.04 MB/s
(base) PS D:\2-uBlock>
```

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

图 10 吞吐量程序输出（左）和官方文档吞吐量（右）

为了减小偶然误差，我们重复 20 次实验，得到平均数据：

[AES-EVP (OpenSSL)] 平均吞吐量: 3589.89 MB/s

[AES-Legacy] 平均吞吐量: 127.57 MB/s

[uBlock-128] 平均吞吐量: 231.66 MB/s

（更多数据见附件《吞吐量数据》）

4.2.2 数据分析及成因简析

其中，官方测试 1MB 数据 uBlock 加密吞吐量为 1809 Mbps，换算为 226.125 MB/s，与我测试的吞吐量大致吻合，说明性能测试程序大致正确。

4.2.2.1 性能对比

将实验数据整理如下表：

加密方式	平均值	最大值	最小值	波动范围	标准差
AES-EVP (OpenSSL)	3578.8	3884.45	3193.77	~690	169.2
AES-Legacy	127.4	145.73	115.49	~30	7.1
uBlock-128	231.6	239.41	222.66	~17	4.3

表 8 三种加密方式的性能数据表

性能排序： AES-EVP ≫ uBlock-128 > AES-Legacy

差距倍数：

AES-EVP 是 uBlock-128 的 15.5 倍

AES-EVP 是 AES-Legacy 的 28.1 倍

uBlock-128 是 AES-Legacy 的 1.8 倍

4.2.2.2 数据波动分析

数据波动性分析如下：

加密方式	波动率（(Max-Min)/Avg）	稳定性评价
AES-EVP	19.3%	波动较大，受系统负载影响显著
AES-Legacy	23.5%	中等波动，函数调用开销敏感
uBlock-128	7.3%	高度稳定，算法优化效果显著

表 9 波动性分析表

4.2.2.3 原因分析

在加密吞吐量上，AES-EVP 保持速率的绝对优势，主要取决于硬件加速（AES-NI）和 openssl 的优化，比如流水线、并行加密等等，难以被超越。

而传统的 AES-Legacy 加密吞吐量最低，主要是因为调用的函数 AES_encrypt 每次只能加密 128 位，即 16 字节数据，在函数调用上花费极大。同时，这种 aes 加密，未使用 AES-NI，纯是软件代码，软件查表（如 T 表），效率低下。

设计的 uBlock，在函数实现上，调用 SIMD/SSE 指令，提升了单块加密的速率，效率明显高于传统软件实现。

在波动性上，AES-EVP 波动较大，可能是因为 CPU 被后台任务抢占资源，受 CPU 影响较大。

AES-Legacy 受函数调用开销的影响，波动也比较大；uBlock 高度稳定，并行结构、模块化、简单的 S 盒和线性变换，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 图像加密实验

实验目标为：使用 uBlock 加密算法实现 ECB 和 CBC 密码算法工作模式，且以上两种工作模式加密一张图，验证加密正确性。

5.1 ECB 和 CBC 工作模式实现

我们为了密钥工作模式的实现，利用 uBlock 实现程序 uBlock_128.cpp 设计出 uBlock 加密库 uBlock_128.h 便于调用函数，后续分别讲解两种加密模式的具体实现。

5.1.1 ECB 工作模式

5.1.1.1 ECB 简要介绍

ECB 模式，即电子密码本模式（Electronic Codebook），是最基础的分组密码模式。

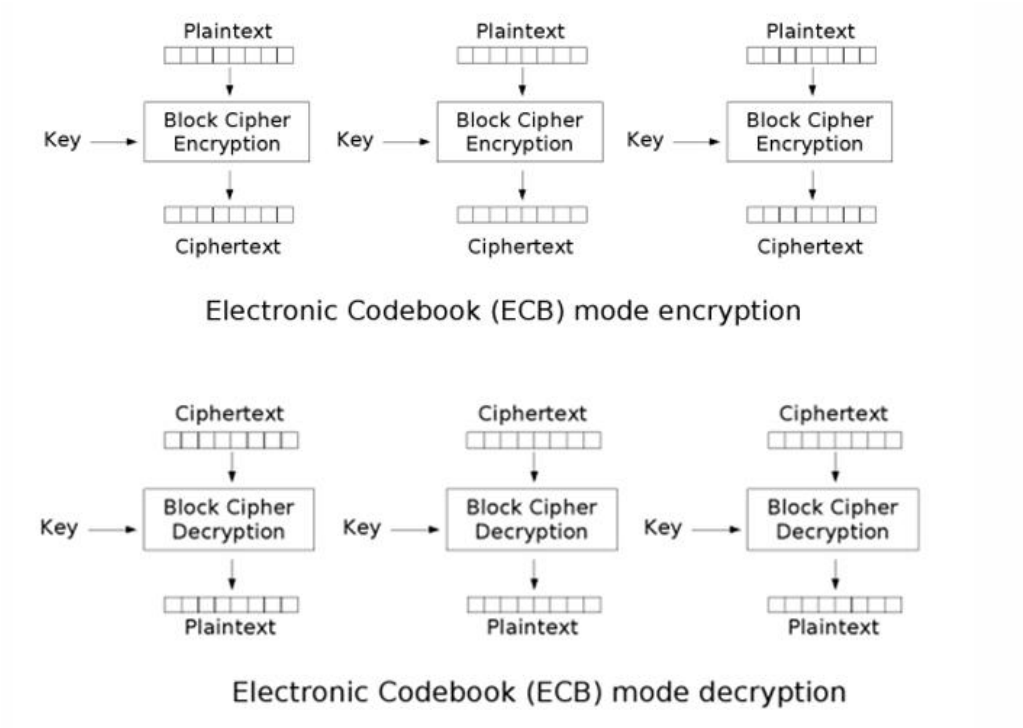


图 11 ECB 工作模式示例图

明文被划分成固定大小的块（如 AES 的 128 位），每个块独立加密，生成对应的密文块，整个加密过程无需初始向量（IV）、计数器或其他上下文，仅依赖密钥。

数学表达式为

$$C_i = E_k(P_i)$$

其中，C_i是第 i 个密文块，P_i是第 i 个明文块，E_k是分组加密算法，如 AES 和此文中的 uBlock。

对于 ECB 加密，显然优点是实现简单，可以通过多线程、SSE 等并行加速，但缺点也很明显，相同明文块加密之后生成相同的密文块，导致图像加密后仍然保留轮廓，无法抵抗重放攻击等等。所以在之后的图像加密实验中，我们可以预想到加密出来的图像，仍然保有戴胜的特征。

5.1.1.2 ECB 工作模式实现程序流程图

ECB 实现程序见附件 uBlock_bmp_ecb.cpp。

程序的主要功能是实现读取 bmp 文件，利用 uBlock 加密和解密文件。

主要流程是：

加密过程：读取 bmp 文件，读取文件头后创建加密图片写入文件头，每 16 字节加密文件；

解密过程：读取加密文件，跳过文件头，每 16 字节解密文件。

详细程序流程见下图：

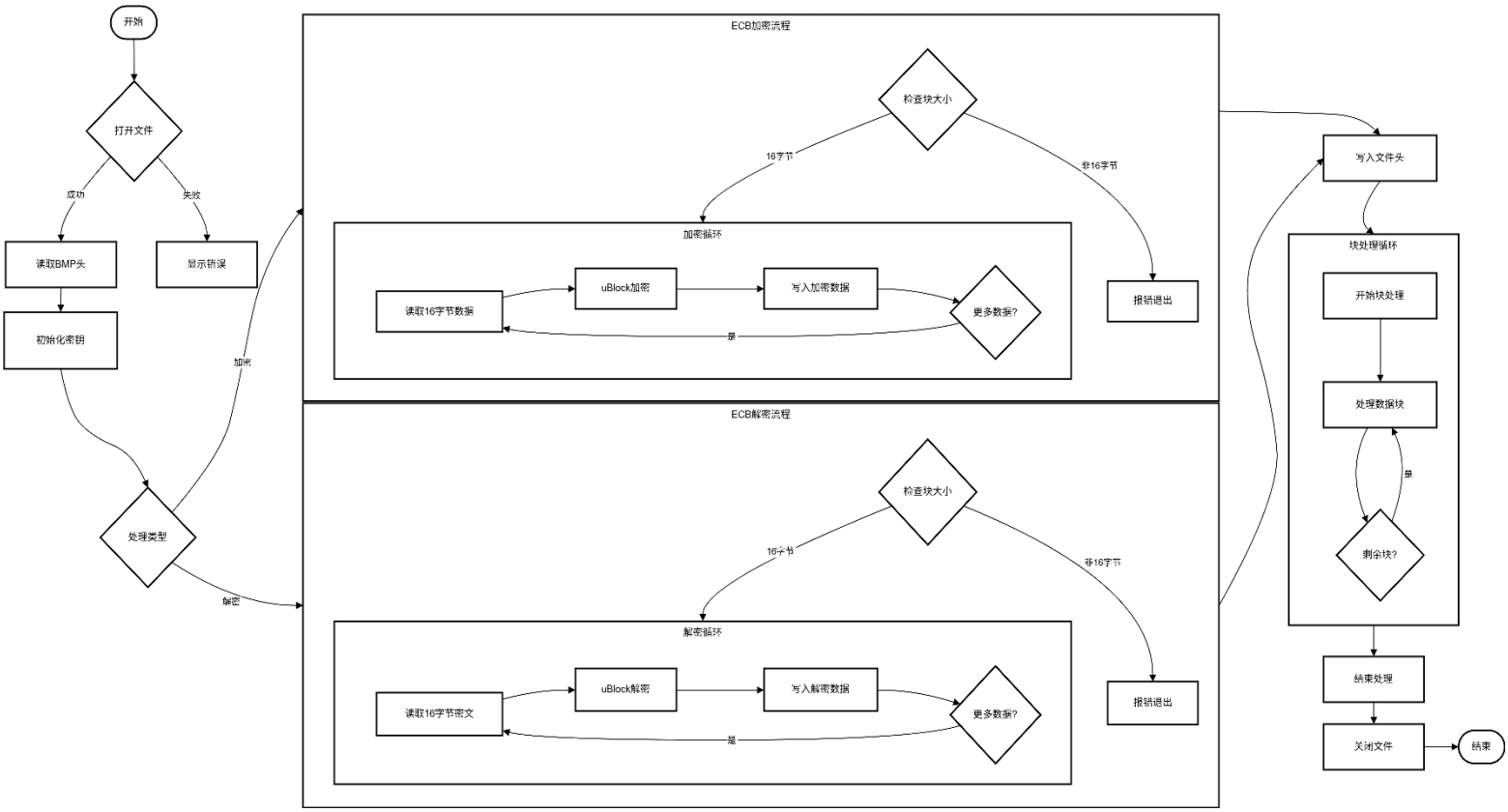


图 12 ECB 工作模式实现程序流程图

具体函数实现见附件。

5.1.2 CBC 工作模式

5.1.2.1 CBC 简要介绍

CBC 模式，即密码块链接（Cipher Block Chaining），是一种广泛使用的分组加密模式。CBC 模式中引入块间的依赖关系和初始向量 IV，解决了 ECB 模式部分安全缺陷。

在 CBC 加密过程中，会随机生成一个与块大小相同的初始向量（如 AES-128 的 16 字节），作为初始向量 IV 与第一块明文块异或；每个明文块在加密前，先与前一个密文块进行按位异或操作；加密后的密文块将作为下一个块的 XOR 输入，形成链式依赖。

数学表达为：

$$C_0 = IV$$

$$C_i = E_k(P_i \oplus C_{i-1}) \quad (i \geq 1)$$

在 CBC 解密过程中，每个密文块解密后，与前一个密文块进行 XOR，恢复原始明文，用公式表达为

$$P_i = D_k(C_i) \oplus C_{i-1}$$

CBC 的优点有很多，比如说隐藏明文格式，避免 ECB 模式泄露的问题，在图片加密中，我们观察 CBC 模式加密后的图片，应该看不出轮廓。

CBC 的缺点也很明显，首先是串行加密，加密过程需按顺序处理块，无法并行化（但解密可并行）。其次是 IV 管理问题，IV 必须随机且不可重复。还有填充漏洞风险，若填充验证不当（如 Padding Oracle 攻击），可能被利用破解密钥。

CBC 与 ECB 的对比如下：

特性	ECB（电子密码本）	CBC（密码块链接）
加密方式	每个明文块独立加密，无块间依赖	当前明文块与前一个密文块异或后再加密，形成链式依赖
初始向量（IV）	不需要	必须使用随机且不可预测的 IV
并行加密	支持（块间无依赖）	不支持（加密需串行处理，但解密可并行）
安全性	低（相同明文生成相同密文，导致模式泄露）	中（隐藏明文模式，但需防范 Padding Oracle 攻击）
错误传播	错误仅影响当前块	错误影响当前块和下一块
填充需求	需要填充（除非明文长度正好为块大小的整数倍）	必须填充
适用场景	加密随机数据（如密钥）、单块加密、硬件受限环境	通用数据加密（如文件、数据库）、旧系统兼容
典型应用	加密单个密钥、无结构数据	TLS 1.2、旧版文件加密工具
性能	高（支持并行处理）	中（加密过程串行）

表 10 ECB 与 CBC 对比表

5.1.2.2 CBC 工作模式实现程序流程图

CBC 实现程序见附件 uBlock_bmp_cbc.cpp。

程序的主要功能是实现读取 bmp 文件，利用 uBlock 加密和解密文件。

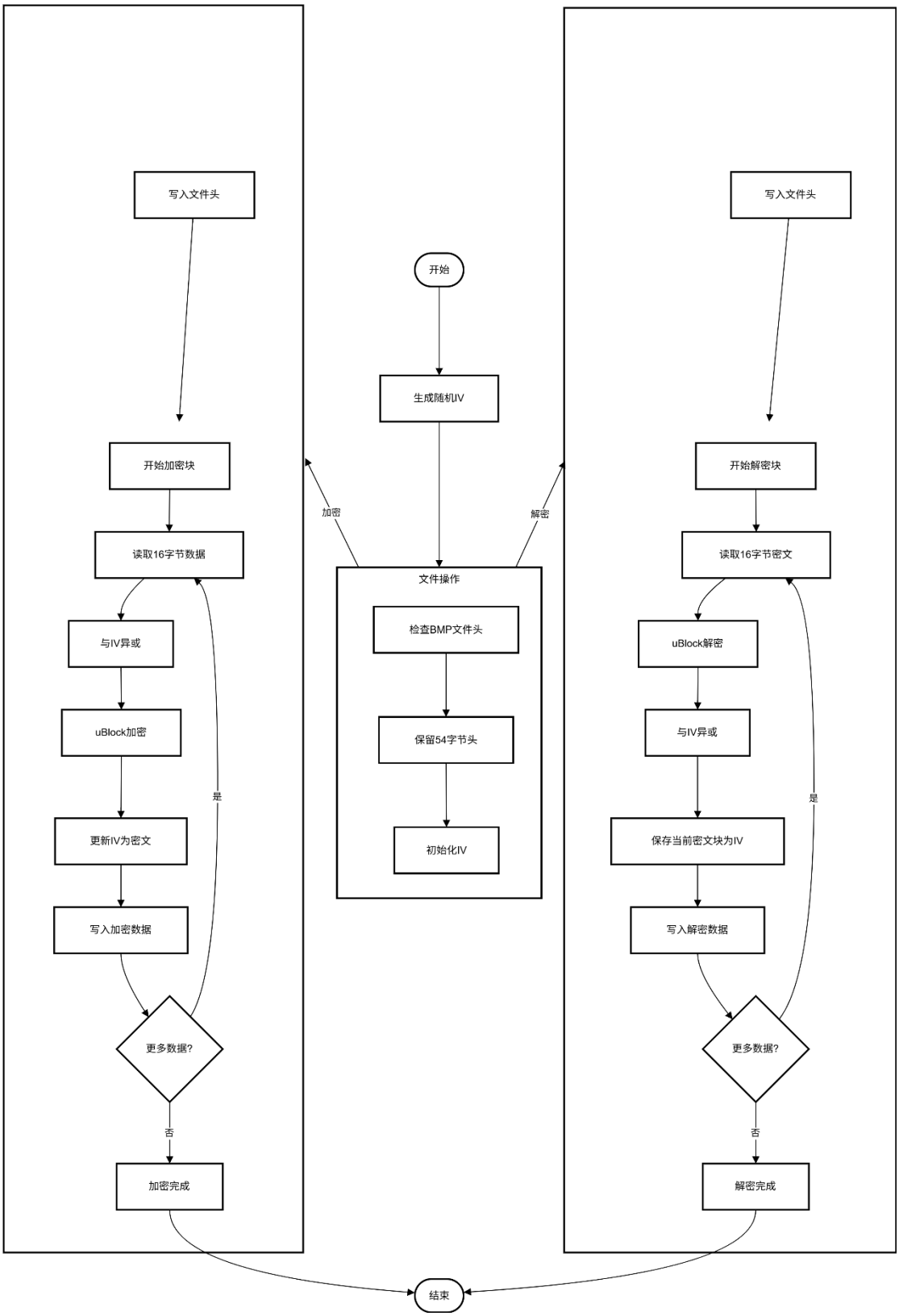


图 13 CBC 工作模式实现程序流程图

流程与 ECB 大致相同，具体函数实现见附件。

5.2 图像加密处理

5.2.1 Bmp 图片格式介绍

Bmp 图片，全称 Bitmap(位图)文件扩展名为.bmp，是 Windows 操作系统中的标准图像文件格式，能够被多种 Windows 应用程序所支持。这种格式的特点是包含的图像信息较丰富，几乎不进行压缩。Bmp 文件由四个部分组成按顺位排列为：文件头、信息头、调色板、像素数据。

文件头（BITMAPFILEHEADER），长度为 14 字节，关键字段如下：

字段名	大小（字节）	描述
文件类型	2	固定为 `BM`（ASCII 码）
文件大小	4	整个文件的大小（字节）
保留字段	4	通常为 0
像素数据偏移量	4	像素数据起始位置（字节）

表 11 文件头关键字段表

信息头（BITMAPINFOHEADER），长度为 40 字节，关键字段如下：

字段名	大小	描述
信息头大小	4	信息头结构长度（固定为 `40` 字节，对应十六制 `0x28`）
图像宽度	4	以像素为单位的宽度值（正值为从左到右，负值表示镜像存储）
图像高度	4	以像素为单位的高度值（正值为从下到上存储，负值为从上到下）
位面数	2	固定为 1（历史遗留字段，无实际意义）
每像素位数（BPP）	2	颜色深度： `1`（单色）、`4`（16 色）、`8`（256 色）、`24`（真彩色）、`32`（带透明）

字段名	大小	描述
压缩方式	4	压缩标识： 0（无压缩）、1（RLE8）、2（RLE4）、3（位域模式）
像素数据大小	4	像素数据的总字节数（需按 4 字节对齐计算）
水平分辨率	4	水平方向每米的像素数（通常为 0，表示未指定）
垂直分辨率	4	垂直方向每米的像素数（通常为 0）
调色板颜色数	4	实际使用的调色板颜色数（若为 0，则默认使用最大颜色数：2^{BPP}）
重要颜色数	4	关键颜色数量（通常为 0，表示所有颜色均重要）

表 12 信息头关键字段表

调色板（Color Table），一般 4 字节。调色板数据根据 BMP 版本的不同而不同，Palette N * 4 byte 调色板规范。对于调色板中的每个表项，这 4 个字节用下述方法来描述 RGB 的值：

1 字节用于蓝色分量
1 字节用于绿色分量
1 字节用于红色分量
1 字节用于填充符（设置为 0）

像素数据（Pixel Data），存储方式为：从左到右，从上到下，每行像素数据需要按照 4 字节对齐。

5.2.2 ECB 图像加密和验证

执行 uBlock_bmp_ebc.cpp，密钥设置为 key[16] = { 0x01 , 0x23 , 0x45 , 0x67 , 0x89 , 0xAB , 0xCD , 0xEF , 0xFE , 0xDC , 0xBA , 0x98 , 0x76 , 0x54 , 0x32 , 0x10};加载图片加密如下：

图 14 ECB 模式加密图像（左）和解密图像（右）

我们看出图片的特征/轮廓在 ECB 模式下得以保存，可以明显的看出中间的一道树枝。

5.2.3 CBC 图像加密和验证

执行 uBlock_bmp_cbc.cpp，密钥设置为 key[16] = { 0x01 , 0x23 , 0x45 , 0x67 , 0x89 , 0xAB , 0xCD , 0xEF , 0xFE , 0xDC , 0xBA , 0x98 , 0x76 , 0x54 , 0x32 , 0x10};加载图片加密如下：

图 15 CBC 模式加密图像（左）和解密图像（右）

我们可以看出，在 CBC 模式加密下，所有图片轮廓得以隐藏，我们看不出图片原本的样子。

6 结论

6.1 uBlock 的创新性和特色

uBlock 算法创新点丰富，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。

算法特色鲜明，适应性强，易于侧信道防护，乘法复杂度低。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。

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（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。

6.2 实验心得

uBlock 软件实验收获颇丰，从程序环境配置，程序流程图书写，程序头文件引用，硬件加速技术到分组密码加解密流程，轮密钥生成与顺序，硬件加速和软件 T 表，全方面全流程系统的了解了一种密码算法是如何从理论到部署。

从理论层面，uBlock 是轻量级的高校的分组密码算法，设计 PX 结构让安全性能和实现性能双赢，相比于传统的 SP 结构，利用 SIMD 指令进行硬件加速。同时实验表示，uBlock 算法可抗多种密码学攻击，比如说差分分析和线性分析。同时，这次实验让我加强了分组密码体制和工作模式的理论知识，为以后进一步的学习奠定坚实基础。

在软件实现上面，第一次基于源代码全流程设计一种分组加密，更深入的了解了 openssl 和 evp 加速，更熟练的掌握 vscode。

在我测试 uBlock 吞吐量时，实验数据表明，uBlock 算法在吞吐量和稳定性都优于传统只有软件实现的 AES，但是硬件加速（AES-NI）后的 AES 还是保持绝对领先，说明 uBlock 的实现还可以再优化。

在我设计图像加密任务中，再次了解了 ECB 和 CBC 两种加密模式，ECB 模式虽然加密效率高，但是不能隐藏图像本来的结构；CBC 虽能有效的掩盖图像特征，有更高的安全性，但是不能并行加密，每一块密文都依赖于前面的密文块，如果有一块出错，后面的密文块都会出错。

总之，我了解了 uBlock 加解密理论、算法实现和性能测试的全流程，充分的体验了密码工程中理论与工程实践的相辅相成。尤其是现在，随着移动设备的缩小和物联网技术的发展，我们需要一种轻量级的算法，uBlock 乘法复杂度低，是一种优秀的轻量级算法。同时我也明白，在以后设计一种密码算法时，不仅要充分考虑软硬件实验的需要，更要去兼顾资源受限设备时的使用场景。自主知识产权的密码设计是国家安全的必需品，uBlock 的设计让我意识到我国密码学研究原创性成果在大国竞争中的重要意义。本篇研究报告为我在密码学研究领域奠定了基础，我也希望继续在这一领域深入研究。