面向移动设备的国密SM2高效实现研究

张吉鹏[1],黄军浩[2,3],于　璇[1],刘　哲[1,4]

（1. 南京航空航天大学计算机科学与技术学院,江苏南京 211106;2. 香港浸会大学,香港 999077;
3. 北京师范大学-香港浸会大学联合国际学院,广东珠海 519087;4. 之江实验室,浙江杭州 311101）

摘　要： SM2的优化实现在x86-64架构上已经得到了充分的研究,但在ARMv8-A架构上的优化仍不充分,为此本工作提出了以下优化方案:针对SM2的模p与模n乘法/平方运算,充分利用p与n的数值特点优化了蒙哥马利模乘;针对模p与模n求逆运算,推导并实现了更快的基于费马小定理的模逆算法;针对固定点与非固定点标量乘法,分别实现了宽度为7与5的窗口算法;针对签名生成过程中s的计算,用一个模n加/减法替换一个模n乘法. 将上述优化技术集成到OpenSSL(3.0.0-beta1)中后,在华为云鲲鹏920计算平台上的测试表明,SM2签名性能提升8.7倍;SM2验签性能提升3.5倍. 在移动设备树莓派4平台上,SM2的签名性能提高9.7倍;验签性能提高3.4倍.

关键词： 椭圆曲线密码;ARMv8-A平台;SM2优化实现;有限域运算;模逆运算

基金项目： 国家重点研发计划(No.2020AAA0107703);国家自然科学基金(No.62132008);霍英东教育基金(No.171057);江苏省杰出青年基金(No.BK20220075)

中图分类号： TP309　　**文献标识码：** A　　**文章编号：** 0372-2112(2023)12-3437-07

电子学报URL:http://www.ejournal.org.cn　　**DOI:**10.12263/DZXB.20221419

Research on Efficient Implementation of SM2 for Mobile Devices

ZHANG Ji-peng[1], HUANG Jun-hao[2,3], YU Xuan[1], LIU Zhe[1,4]

（1. *College of Computer Science and Technology*, *Nanjing University of Aeronautics and Astronautics*, *Nanjing*, *Jiangsu* 211106, *China*;
2. *Hong Kong Baptist University*, *Hong Kong* 999077, *China*;
3. *BNU-HKBU United International College*, *Zhuhai*, *Guangdong* 519087, *China*;
4. *Zhejiang Laboratory*, *Hangzhou*, *Zhejiang* 311101, *China*）

Abstract: SM2 has been fully studied on x86-64 architecture, but its optimization on ARMv8-A architecture is inadequate. In this work, we propose the following optimizations to fill this gap: for the modular multiplication/squaring of p and n in SM2, we optimize Montgomery modular multiplication/squaring by leveraging the numerical characteristics of p and n; for the modular inversion of p and n in SM2, we derive and implement a faster modular inversion algorithm based on Fermat´s little theorem; for fixed-point and unknown-point scalar multiplication, we implement window algorithms with a window width of 7 and 5, respectively; for the calculation of s during the signature generation process, we replace a modular multiplication of n with a cheaper modular addition/subtraction of n. After integrating the optimizations mentioned above into OpenSSL (3.0.0-beta1), the benchmark on the HUAWEI Cloud Kunpeng 920 computing platform shows that the performance of SM2 signature generation is accelerated by 8.7 times; the performance of SM2 signature verification is accelerated by 3.5 times. Meanwhile, on the mobile device Raspberry Pi 4 platform, the performance of SM2 signature generation is accelerated by 9.7 times; the performance of SM2 signature verification is accelerated by 3.4 times.

Key words: elliptic curve cryptography; ARMv8-A platform; optimized implementation of SM2; finite field operation; modular inversion operation

Foundation Item(s): National Key Research and Development Program (No.2020AAA0107703); National Natural Science Foundation of China (No.62132008); Fok Ying Tung Foundation (No.171057); Jiangsu Outstanding Youth Fund (No.BK20220075)

收稿日期:2022-12-21;修回日期:2023-07-31;责任编辑:李勇锋

1 引言

椭圆曲线密码(Elliptic Curve Cryptography,ECC)由于其密钥长度短、计算效率高等优势[1],逐渐取代RSA成为网络协议中应用最为广泛的公钥密码算法. 当前国际上使用最广泛的ECC标准有NIST P-256[2]、X25519[3]和Ed25519[4].

2010年国家密码管理局公布了一系列商用密码算法标准,其中包括SM2椭圆曲线公钥密码算法标准. SM2采用与NIST P-256类似的短Weierstrass类型的椭圆曲线($y^2=x^3+ax+b$),并且参数选取具有高度自主可控性.

在移动互联网和云计算产业高度发达的当下,基于ARMv8-A架构的处理器由于其优异的功耗表现成为移动设备的主流处理器. 该架构处理器还被用于设计服务器与超级计算机,如华为鲲鹏系列服务器[5]和日本超级计算机"富岳"[6]. 目前SM2在ARMv8-A架构上的研究仍不充分,为此本工作提出了一系列的优化方案以提升SM2在ARMv8-A架构上的性能表现.

1.1 相关工作

Bernstein等人[7]在ARM32-NEON平台上,使用NEON指令优化了Curve25519算法. Faz-Hernández等人[8]使用AVX2指令优化了Curve25519和Curve448,他们的优化与文献[7]的思路类似,也是利用了Curve25519模数的特性. Adalier等人[2]针对NIST P-256曲线在x86-64平台做了优化实现,并表明蒙哥马利约简优于快速约简和Barrett约简. Koc等人[9]分析了五种蒙哥马利模乘实现方法,分别是SOS、CIOS、FIOS、FIPS和CIHS. 他们的实验表明将多精度乘法和蒙哥马利约简相结合执行的CIOS算法性能最优. Mai等人[10]在x86-64平台上优化了SM2数字签名算法. Gueron等人[11]针对NIST P-256曲线做了优化,他们利用了模数的数值特征优化了蒙哥马利模乘. 针对固定点标量乘法,他们使用窗口宽度$w=7$的窗口算法进行优化实现. 而针对非固定点标量乘法,则使用了$w=5$的窗口算法.

目前OpenSSL和GmSSL库中提供了SM2的C语言实现和x86-64汇编实现,但SM2在ARMv8-A架构上的研究仍不充分. 据我们所知,目前没有公开的论文或代码库提供SM2在ARMv8-A架构上的汇编优化实现.

2 背景知识

2.1 基本符号

在展开介绍ECC之前,本节对下文所用的基本符号进行概括. 小写字母如椭圆曲线系数$a_1 \sim a_6$、椭圆曲线点坐标x/y和标量k均为有限域F_P内的元素,p为有限域运算的模数,n为ECC基点的阶. 小写字母$\tilde{a}$和$\tilde{b}$表示元素a和b在蒙哥马利域中所对应的元素. 大写字母如G、P和Q表示椭圆曲线上的点,椭圆曲线点在仿射坐标中可表示为(x,y),而在雅可比投影坐标中则用(X,Y,Z)表示. 椭圆曲线所有点的集合用E/F_P表示. 大写字母W表示机器字长,小写字母w表示计算标量乘法的窗口算法窗口宽度.

2.2 有限域运算

有限域F_P上的核心运算包括模加/减、模乘/平方和模逆运算,其中模加/减运算较为简单,但涉及条件加/减p的运算. 而条件加/减法会使ECC遭受侧信道攻击的威胁,因此常采用掩码技术实现条件加/减运算以抵御简单能量攻击. 模乘/平方运算使用最为频繁,因此模乘/平方的优化对ECC整体运行效率的提升至关重要.

模乘/平方运算的实现可分为两个步骤:(1)乘法/平方运算,通常使用教科书乘法(schoolbook multiplication)进行实现,乘数的位数大于机器字长时称为大整数或多精度乘法/平方运算;(2)模约简运算,常用恒定时间的蒙哥马利约简算法或基于模数数值特征的快速约简算法进行实现. 对于SM2,快速约简算法利用等价关系$2^{256}=2^{224}+2^{96}-2^{64}+1 \bmod p$将两个大整数乘积中权重高于$2^{256}$的字转换成低权重字的加/减运算. 其只适用于具有良好数值特性的模数. 而蒙哥马利约简算法则是使用移位运算替换除法运算. 令$\beta=2^{W_s}$,其中W为机器字长,s为大整数字数,且要求$\beta>p$,$\mathrm{GCD}(\beta,p)=1$. 假设大整数$a,b \in F_P$的乘积为t. 该算法计算$c=t\beta^{-1} \bmod p$,其计算原理为$c=\left(t+(tp' \bmod \beta)p\right)/\beta$,其中$p'=-p^{-1} \bmod \beta$,除以$\beta$与右移$W_s$位等价. 最后还需要执行一次条件减$p$运算.

在计算模乘运算时,通常将多精度乘法与模约简结合起来执行,如文献[9]中的CIOS方法,其64位实现如算法1所示. 该算法将蒙哥马利约简嵌入到多精度乘法的外循环中,每次循环对低64位字进行约简,循环结束后即可完成对乘积t的约简. 最后需要对t进行条件减p使结果处于$[0,p)$范围内.

此外,恒定时间的模逆运算常使用费马小定理($a^{-1}=a^{p-2} \bmod p$)进行实现.

2.3 椭圆曲线点运算

在仿射坐标(椭圆曲线点表示为(x,y))下计算点加/倍点涉及复杂的模逆运算,因此通常使用三坐标的雅可比投影坐标(X,Y,Z),$Z \neq 0$来表示椭圆曲线点. 雅可比投影坐标点(X,Y,Z)与仿射坐标点$(X/Z^2, Y/Z^3)$等价. 雅可比投影坐标下的短Weierstrass等式为$Y^2=X^3+aXZ^4+bZ^6$. 将$x=X/Z^2$,$y=Y/Z^3$代入仿射坐标下的点加和倍点计算公式即可得到雅可比投影坐标下的点

算法1　蒙哥马利模乘CIOS方法

输入: 有限域 F_P 中的元素 $a=\sum_{i=0}^{3}a_i 2^{64i}$, $b=\sum_{i=0}^{3}b_i 2^{64i}$, 模数 p, 常量 $p'=-p^{-1} \bmod 2^{256}$, 常量 $\beta=2^{256}$

输出: $t=ab\beta^{-1} \bmod p$

1. FOR $i \leftarrow 0$ TO 3 DO
2. 　$C \leftarrow 0$;/*C用于保存进位*/
3. 　FOR $j \leftarrow 0$ TO 3 DO
4. 　　$(C,S) \leftarrow t_j + a_j b_i + C$;/*$t_0$~$t_5$初始为0*/
5. 　　$t_j \leftarrow S$;
6. 　END FOR
7. 　$(C,S) \leftarrow t_4 + C$;
8. 　$t_4 \leftarrow S, t_5 \leftarrow C$;/*$t \leftarrow b_i a + t$*/
9. 　$m \leftarrow t_0 p_0' \bmod 2^{64}$;/*$p_0' \leftarrow p' \bmod 2^{64}$*/
10. 　$(C,S) \leftarrow t_0 + mp_0$;
11. 　FOR $j \leftarrow 1$ TO 3 DO
12. 　　$(C,S) \leftarrow t_j + mp_j + C$;
13. 　　$t_{j-1} \leftarrow S$;
14. 　END FOR
15. 　$(C,S) \leftarrow t_4 + C$;
16. 　$t_3 \leftarrow S$;
17. 　$t_4 \leftarrow t_5 + C$;
18. END FOR
19. IF $t \geqslant p$ THEN $t \leftarrow t-p$;
20. RETURN t.

加/倍点计算公式(参考文献[1]中的式3.13和式3.14).雅可比投影坐标下的点加/倍点运算不需要计算模逆,具有较高的计算效率.

2.4　标量乘运算

标量乘法运算是ECC的安全基础与核心运算,通常采用窗口算法对标量乘进行加速,见算法2. 该算法先将一定数量的椭圆曲线点预计算并保存在预计算表中,每次扫描标量的w个比特,根据w比特的值从预计算表中取椭圆曲线点并进行运算. 窗口宽度w的选择涉及时间与空间,即计算开销与预计算表存储空间之间的权衡.

算法2　标量乘法-窗口算法,窗口宽度为w

输入: 标量k,其被划分为l个w比特,表示为$\sum_{i=0}^{l-1}k_i 2^{wi}$,$s$为$k$的比特长度,$l=\lceil s/w \rceil$,椭圆曲线点$P \in E(F_P)$

输出: 标量乘法结果$Q=kP$

预计算 table$[i] \leftarrow iP, i=0,\cdots,2^w-1, i=0$时 table$[0] \leftarrow \infty$;
$Q \leftarrow \infty$;
FOR $i \leftarrow l-1$DOWN TO 0 DO
　$Q \leftarrow 2^w Q$;
　$Q \leftarrow Q + \text{table}[k_i]$;
END FOR
RETURN Q;

2.5　SM2数字签名协议

SM2数字签名协议包含三个核心步骤:密钥生成、签名生成和签名验证,其具体计算过程见国家密码管理局发布的标准文献[12].

2.6　ARMv8-A平台介绍

ARMv8-A架构是ARM公司面向高性能应用设计的64位架构,ARMv8-A提供了一组64位指令集(A64),其支持一系列64位的算术和逻辑运算,并支持较为丰富的乘法运算,比如乘法(MUL)、乘法取负(MNEG)、乘加(MADD)和乘减(MSUB)等指令. 两个64位数相乘需要使用两条指令({S,U}MULH与{S,U}MULL)才能得到完整的128位结果. ARMv8-A架构提供了31个通用寄存器x_0~x_{30}. ARMv8-A还支持128位单指令多数据(SIMD)指令集NEON,但是其只支持32位乘法运算,并且不支持进位/借位处理,所以在ARMv8-A平台上很难将NEON用于优化ECC.

3　优化方案

本文针对SM2的优化主要包括:(1)针对模p与模n乘法/平方运算,本文充分利用SM2模数p和阶n的数值特征优化了蒙哥马利模乘运算;(2)针对模p与模n求逆运算,本文推导并实现了更快的基于费马小定理的模逆运算;(3)针对固定点与非固定点标量乘法,本文分别实现了宽度为7与5的窗口算法;(4)针对签名生成算法中s的计算,本文通过公式等价变换,将一个模n乘法运算替换成一个模n加/减法.

本文的优化实现是开源的,详情见https://github.com/Ji-Peng/openssl_SM2_ARMv8-A.

3.1　模乘优化

3.1.1　模p时的蒙哥马利模乘优化

与文献[11]中优化NIST P-256曲线所采用的思路类似,本文根据模数的数值特征,通过变换模数的表示将约简部分的乘法运算等价变换为加/减法,从而减少计算开销. SM2模数$p=2^{256}-2^{224}-2^{96}+2^{64}-1$的十六进制形式为

$$p_0=FFFFFFFFFFFFFFFF$$
$$p_1=FFFFFFFF00000000$$
$$p_2=FFFFFFFFFFFFFFFF$$
$$p_3=FFFFFFFEFFFFFFFF$$
$$p=p_0+p_1 2^{64\cdot 1}+p_2 2^{64\cdot 2}+p_3 2^{64\cdot 3}$$

考虑算法1第10行中的mp$_0$: mp$_0=m\left(2^{64}-1\right)=m2^{64}-m$,其中$m2^{64}$等价于将$m$累加至权重为$2^{64}$的项中,$-m$等价于对权重为$2^0$的项减$m$,这样可将乘法转换为运行效率更高的加/减法.

基于上述观察,将SM2模数等价变换为:$p=\left(2^{64}-\right.$

$2^{32})2^{192}+\left(1-2^{32}\right)2^{64}+(-1)$，即 $p_0=-1, p_1=1-2^{32}, p_2=0, p_3=2^{64}-2^{32}$. 从而可以将算法1第10、12行的 mp_j 转换为

$mp_0=-m$；$mp_1=m\left(1-2^{32}\right)=-(m\gg 32)2^{64}+m-(m\ll 32)$；$mp_2=0$；$mp_3=m(2^{64}-2^{32})=(m-(m\gg 32))2^{64}-(m\ll 32)$

因此，算法1第10、12行的乘法运算 mp_j 全部被转换为了加/减法运算，其中算法1第10~16行的运算变换为

$$(C,S)=t_0-m=t_0-t_0=0$$
$$(C,S)=t_1+m-(m\ll 32), t_0=S$$
$$(C,S)=t_2-(m\gg 32)+C, t_1=S$$
$$(C,S)=t_3-(m\ll 32)+C, t_2=S$$
$$(C,S)=t_4+m-(m\gg 32)+C,\ t_3=S$$

在SM2中算法1第9行用到的 $p_0'=1$，可节省一个乘法运算[13]，同时 $t_0-m=t_0-t_0=0$ 运算也可忽略. 模平方的约简部分采用相同的优化思路. 算法3给出了约简部分ARMv8-A汇编实现伪代码.

算法3 蒙哥马利模乘优化(模数为 p)实现中约简运算的ARMv8-A汇编实现(对应算法1第9~17行)

输入:算法1中2~8行的中间乘积:$t=\sum_{i=0}^{5}t_i 2^{64^i}$

输出:约简后的结果:$t=\sum_{i=0}^{4}t_i 2^{64^i}$

MOV m, t_0;/*$m=t_0 p_0{}' \bmod 2^{64}$ ($p_0{}'=1$)*/
LSL ml, m, #32;/*$m\ll 32$*/
LSR mr, m, #32;/*$m\gg 32$*/
ADDS t_0, t_1, m;/*t_1+m*/
ADCS t_1, t_2, 0;/*t_2+C*/
ADCS t_2, t_3, 0;/*t_3+C*/
ADCS t_3, t_4, m;/*t_4+m+C*/
ADCS t_4, t_5, 0;/*t_5+C*/
SUBS t_0, t_0, ml;/*t_0-ml*/
SBCS t_1, t_1, mr;/*t_1-mr-B(B表示借位)*/
SBCS t_2, t_2, ml;/*t_2-ml-B*/
SBCS t_3, t_3, mr;/*t_3-mr-B*/
SBC t_4, t_4, 0;/*t_4-B*/
RETURN $t=\sum_{i=0}^{4}t_i 2^{64^i}$.

3.1.2 模 n 时的蒙哥马利模乘优化

本小节仍基于算法1进行描述，但需将其中的模数 p 替换为阶 n，将 $p'=-p^{-1} \bmod 2^{256}$ 替换为 $n'=-n^{-1} \bmod 2^{256}$. SM2阶 n 的十六进制形式为

$$n_0=53BBF40939D54123$$
$$n_1=7203DF6B21C6052B$$
$$n_2=FFFFFFFFFFFFFFFF$$
$$n_3=FFFFFFFEFFFFFFFF$$
$$n=n_0+n_1 2^{64\cdot 1}+n_2 2^{64\cdot 2}+n_3 2^{64\cdot 3}$$

其中 mn_2 与 mn_3 可使用与3.1.1小节相同的思路进行优化.

算法1第10行的 mn_0 我们只需要计算该乘法的高64 bit结果，原因是其满足下述条件：

$$\left(t_0+m\cdot n_0\right) \bmod 2^{64}$$
$$=\left(t_0+\left(t_0\cdot\left(-n_0{}^{-1} \bmod 2^{64}\right)\right)\cdot n_0\right)$$
$$=\left(t_0-t_0\right)=0 \bmod 2^{64}$$

即乘积中的低64 bit恒为0. 但需要注意的是，此处需要分析两种情况：(1) $t_0=0$ 时，$m=0$，此时 $t_0+m\cdot n_0=0$；(2) $t_0\neq 0$ 时，$t_0+m\cdot n_0$ 的低64bit为0，同时会向高64 bit传递一个进位，形如 $\left(2^{64}-1\right)+1$ 的低64 bit为0，并向高64 bit进位，该进位值记为carry.

算法1第12行计算 mn_1 时，因为 n_1 没有形如 n_2 与 n_3 的数值特征，故需要计算完整的128 bit乘法结果. 综上，算法1第10~16行的运算变换为

$$C=\mathrm{high}\left(m\cdot n_0\right)+\mathrm{carry}$$
$$(C,S)=t_1+m\cdot n_1+C, t_0=S$$
$$(C,S)=t_2-m+C, t_1=S$$
$$(C,S)=t_3+m-(m\ll 32)+C, t_2=S$$
$$(C,S)=t_4+m-(m\gg 32)+C, t_3=S$$

其中 $\mathrm{high}\left(m\cdot n_0\right)$ 表示乘积 $m\cdot n_0$ 的高64 bit结果. 其实现方法与算法3类似.

3.2 模逆优化

给定 $0<a<p$，其中 p 为模数，本文利用费马小定理计算 a 的乘法逆元 $a^{-1}=a^{p-2} \bmod p$. 正如Knuth Donald[14]在其书籍第4.6.3章所述，给定一个指数 e 的加法链，计算 $x^e \bmod p$ 时：平方相当于加法链中的乘2运算，乘法相当于加法链中的加法运算. 因此，模逆运算的效率取决于指数加法链的效率[15]，应尽可能减少计算过程中的模乘/平方运算，或者以模平方运算替换模乘运算(模平方的计算效率高于模乘).

3.2.1 已有优化方案分析

对于模 p 求逆，OpenSSL中的SM2实现采用通用的模逆算法，使用了窗口宽度为4的窗口算法[16]扫描指数 $p-2$. 在不考虑预计算开销的情况下，该方案消耗 $(256/4-1)\times 4=252$ 个模平方与 $256/4-1=63$ 个模乘运算. GmSSL虽然不支持ARMv8-A架构，但是其提供了针对SM2模数的模逆优化实现[17]，该方案消耗256个模平方和27个模乘运算. Zhou Lu等人[18]针对SM2的模数 p 提出了更快的模逆算法，其消耗255个模平方和15个模乘运算.

而对于模 n 求逆，GmSSL没有提供专门的优化方案；OpenSSL对NIST P-256曲线的模阶求逆提供了一个优化方案. NIST P-256曲线的阶与SM2的类似，在计算

$Z^{-1}=Z^{n-2} \bmod n$ 时,$n-2$ 的高 128 bit 大部分为比特 1,具有较好的数值规律便于构造高效的加法链;而低 128 bit 则没有明显的数值规律,不方便构造加法链. 因此,OpenSSL 只为高 128 bit 构造了加法链,而低 128 bit 使用宽度为 4 的窗口算法进行扫描.

3.2.2　模 *p* 求逆优化方案

我们观察到,模 p 求逆运算只需要用在雅可比投影坐标转仿射坐标的过程中,即:

$$(x,y)=(XZ^{-2}, YZ^{-3})$$

从上式可知,模 p 求逆运算主要用于计算 Z^{-2} 和 Z^{-3}. SM2 签名过程中 Z_A 的生成需要使用基点 G 和点 d_AG 的 x 和 y 值,因此需计算对应的 Z^{-2} 和 Z^{-3}. 签名生成算法与签名验证算法的后续运算仅涉及 x 值,因此只需要计算对应的 Z^{-2}. 因此,我们直接计算 $Z^{-2}=Z^{p-3} \bmod p$ 并通过 $(Z^{-2})^2 \cdot Z$ 来构造 Z^{-3}.

本文 Z^{-2} 的计算方法见算法 4,算法注释中 S 表示模平方,M 表示模乘,该算法计算开销为 256 个模平方和 14 个模乘. 与 Zhou Lu 等人 Z^{-1} 的计算开销相比,我们用一个模平方运算替换了一个模乘运算;并且当只需计算 Z^{-2} 时,Zhou Lu 等人的方法还需使用一次模平方运算($Z^{-2}=(Z^{-1})^2$),因此我们的优化方案具有更好的计算效率.

算法 4　基于费马小定理计算 Z^{-2}

输入:Z 满足 $0<Z<p$

输出:$Z^{-2}=Z^{p-3} \bmod p$

1. $Z_2 \leftarrow Z^2 \cdot Z$;/*1$S$+1$M$*/
2. $Z_4 \leftarrow Z_2^{2 \cdot 2} \cdot Z_2$; /*2$S$+1$M$*/
3. $Z_6 \leftarrow Z_4^{2 \cdot 2} \cdot Z_2$; /*2$S$+1$M$*/
4. $Z_{12} \leftarrow Z_6^{2 \cdot 6} \cdot Z_6$; /*6$S$+1$M$*/
5. $Z_{24} \leftarrow Z_{12}^{2 \cdot 12} \cdot Z_{12}$; /*12$S$+1$M$*/
6. $Z_{30} \leftarrow Z_{24}^{2 \cdot 6} \cdot Z_6$; /*6$S$+1$M$*/
7. $Z_{31} \leftarrow Z_{30}^2 \cdot Z$;/*1$S$+1$M$*/
8. $Z_{32} \leftarrow Z_{31}^2 \cdot Z$;/*1$S$+1$M$*/
9. $t \leftarrow (Z_{31}^{2 \cdot 33} \cdot Z_{32})^{2^{32}} \cdot Z_{32}$; /*65$S$+2$M$*/
10. $t \leftarrow (t^{2^{32}} \cdot Z_{32})^{2^{32}} \cdot Z_{32}$; /*64$S$+2$M$*/
11. $t \leftarrow (t^{2^{64}} \cdot Z_{32})^{2^{30}} \cdot Z_{30}$; /*94$S$+2$M$*/
12. $t \leftarrow t^{2^2}$; /*2S*/
13. RETURN t;/*总开销为:256S+14M*/

3.2.3　模 *n* 求逆优化方案

我们的优化方法见算法 5,我们先预计算了 11 个常用值,包括 Z^{0b11}、Z^{0b101}、Z^{0b111} 等(0b 为二进制表示). 然后使用这些预计算值来构建 Z^{n-2}. 该算法的总开销为 $42M+257S$. 相比于 OpenSSL 中的方案(其开销为 $49M+260S$),算法 5 节省了 7 次模乘与 3 次模平方运算.

算法 5　优化版模 n 求逆算法

输入:Z 满足 $0<Z<n$

输出:$Z^{-1}=Z^{n-2} \bmod n$

1. 预计算:$Z^{0b1}, Z^{0b11}, Z^{0b101}, Z^{0b111}, Z^{0b1001}$,
   $Z^{0b1011}, Z^{0b1111}, Z^{0b10101}, Z^{0b11111}$,
   $Z_{x31} \leftarrow Z^{0 \times 7FFFFFFF}, Z_{x32} \leftarrow Z^{0 \times FFFFFFFF}$;
2. $r \leftarrow {Z_{x31}}^{2^{33}} \cdot Z_{x32}$,　$r \leftarrow r^{2^{32}} \cdot Z_{x32}$;
3. $r \leftarrow r^{2^{32}} \cdot Z_{x32}$,　$r \leftarrow r^{2^4} \cdot Z^{0b111}$;
4. $r \leftarrow r^{2^3} \cdot Z^{0b1}$,　$r \leftarrow r^{2^{11}} \cdot Z^{0b1111}$;
5. $r \leftarrow r^{2^5} \cdot Z^{0b1111}$,　$r \leftarrow r^{2^4} \cdot Z^{0b1011}$;
6. $r \leftarrow r^{2^5} \cdot Z^{0b1011}$,　$r \leftarrow r^{2^3} \cdot Z^{0b1}$;
7. $r \leftarrow r^{2^7} \cdot Z^{0b111}$,　$r \leftarrow r^{2^5} \cdot Z^{0b11}$;
8. $r \leftarrow r^{2^9} \cdot Z^{0b101}$,　$r \leftarrow r^{2^7} \cdot Z^{0b10101}$;
9. $r \leftarrow r^{2^5} \cdot Z^{0b10101}$, $r \leftarrow r^{2^5} \cdot Z^{0b111}$;
10. $r \leftarrow r^{2^4} \cdot Z^{0b111}$,　$r \leftarrow r^{2^6} \cdot Z^{0b11111}$;
11. $r \leftarrow r^{2^3} \cdot Z^{0b101}$,　$r \leftarrow r^{2^{10}} \cdot Z^{0b1001}$;
12. $r \leftarrow r^{2^5} \cdot Z^{0b111}$,　$r \leftarrow r^{2^5} \cdot Z^{0b111}$;
13. $r \leftarrow r^{2^6} \cdot Z^{0b10101}$, $r \leftarrow r^{2^2} \cdot Z^{0b1}$;
14. $r \leftarrow r^{2^9} \cdot Z^{0b1001}$,　$r \leftarrow r^{2^5} \cdot Z^{0b1}$;
15. return $r \leftarrow Z^{-1} \bmod n$;

3.3　标量乘优化

在 OpenSSL 的 SM2 实现中,固定点与非固定点标量乘均使用蒙哥马利梯子算法进行计算. Gueron Shay 等人将他们对 NIST P-256 曲线的优化实现[11]集成到了 OpenSSL 中[19],我们将他们的优化方案迁移到了 SM2 中. 固定点标量乘法采用宽度为 7 的窗口算法,非固定点标量乘法采用宽度为 5 的窗口算法,标量使用 Booth 编码[20]的变体.

对于固定点标量乘法,共有 $\lceil 256/7 \rceil=37$ 个窗口,故每个窗口包含 $2^6=64$ 个点. 预计算表中的椭圆曲线点采用仿射坐标 (x,y) 形式进行存储,因为仿射坐标-雅可比投影坐标混合加法运算的运行效率最快(参考文献[1]中的算法 3.22). 预计算表可表示为:$\text{Table}[i][j]=2^{7i} \times (j \times G) \bmod p$,其中 $i=0,1,\cdots,36, j=0,1,\cdots,63$,预计算表总大小为 148 KB. 通过上述预计算技术,固定点标量乘法运算只需要 37 次点加运算,无需倍点运算.

非固定点标量乘法的预计算需在算法运行时执行,所以需要减少预计算量. 本文采用窗口宽度为 $w=5$ 的窗口算法,只需预计算 $2^4=16$ 个点,即 $i \times G, i=0,1,\cdots,15$. 通过上述预计算技术,在计算非固定点标量乘法时,预计算开销为 7 个点加和 8 个倍点运算,扫描窗口时需要 52 个点加和 255 次倍点运算,总开销为 59 个点加和 263 个倍点运算.

3.4　其他优化

观察到 SM2 签名生成算法的如下计算:

$$s=\left(\left(1+d_A\right)^{-1}\cdot\left(k-rd_A\right)\right) \bmod n$$

该过程消耗2个模n加/减法、1个模n求逆和2个模n乘法运算．我们对其进行了如下等价变换：

$$s=\left(k+r\right)\cdot\left(1+d_A\right)^{-1}-r \bmod n$$

变换后的计算消耗3个模n加/减法、1个模n求逆和1个模n乘运算，即使用一个模n加/减法替换一个模n乘法．由于模n加/减法运算开销低于模n乘法运算，因此可以进一步提升签名生成算法的运行效率．

3.5 侧信道攻击分析

为抵御侧信道攻击，本文的实现避免了私钥相关的分支操作．在模加/减运算中，使用掩码技术替换条件加/减法运算；在模乘/平方运算中使用恒定时间的蒙哥马利模乘；在模逆运算中，本文使用恒定运行时间和固定执行流程的费马小定理求逆方案．此外，为抵抗缓存攻击，本文对预计算表的存储和访问做了防护，使得访问每个预计算点都需要访问相同的缓存行，隐藏了缓存行的访问模式．结合以上技术，本文的实现可有效抵抗简单能量分析和缓存攻击这两类侧信道攻击．

4 实验结果

本文的测试环境包括云端和移动端，云端设备为华为云租赁的鲲鹏通用计算增强型（鲲鹏920）服务器，双核CPU，内存为4 GiB，操作系统为64位ARM版Ubuntu 18.04 Server. 移动端设备为树莓派4，其搭载了1.5 GHz的四核ARM Cortex-A72处理器，内存为8 GiB，操作系统为Ubuntu 20.04 Server.

本文所用的时间测试接口与OpenSSL保持一致，模p与模n乘法/求逆运算测试运行了2^{31}次取均值，固定点/非固定点标量乘和签名验签测试运行了2^{23}次取均值．详细实验数据见表1和表2，表格第二列为OpenSSL中SM2的实现性能，第三列为本文优化的SM2实现性能．

表1 SM2核心运算每秒可执行次数对比（$k=10^3$）

平台	函数名	OpenSSL	本文优化	加速比
华为鲲鹏	模p乘法	29 337 k[21]	39 202 k	1.3 倍
	模n乘法	29 337 k[21]	36 665 k	1.2 倍
	模p求逆	76 580	160 158	2.1倍
	模n求逆	73 754	113 124	1.5倍
	固定点标量乘	2 554	77 550	30.4 倍
	非固定点标量乘	2 572	13 406	5.2 倍
树莓派4	模p乘法	6 086 k	11 434 k	1.9 倍
	模n乘法	6 086 k	7 880 k	1.3 倍
	模p求逆	16 931	51 568	3.0倍
	模n求逆	16 647	31 845	1.9倍
	固定点标量乘	714	22 032	30.9 倍
	非固定点标量乘	717	3 886	5.4 倍

表2 签名/验签每秒可执行次数对比

平台	函数名	OpenSSL	本文优化	加速比
华为鲲鹏	签名生成	2 413	21 102	8.7 倍
	验证签名	2 526	8 822	3.5 倍
树莓派4	签名生成	675	6 534	9.7 倍
	验证签名	777	2 646	3.4 倍

5 结论

本文给出了一种SM2在ARMv8-A架构上的优化方案，其中针对有限域模乘运算，本文根据SM2模数p和基点阶n的数值特征优化了蒙哥马利模乘；针对模逆运算，提出了更快的基于费马小定理的模p与模n求逆方法；针对固定点与非固定点标量乘法运算，使用了窗口算法和预计算技术进行优化，使其性能得到了较大幅度的提升．最终实验结果表明，在华为鲲鹏920和树莓派4平台上，相较于OpenSSL中的SM2实现，本文的优化后的签名生成算法性能分别提升8.7倍和9.7倍，验签性能分别提升3.5倍和3.4倍，使得SM2在ARMv8-A架构上的性能得到了充分的发挥．

参考文献

[1] HANKERSON D, MENEZES A J, VANSTONE S. Guide to Elliptic Curve Cryptography[M]. Berlin: Springer Science & Business Media, 2006.

[2] ADALIER M, TEKNIK A. Efficient and secure elliptic curve cryptography implementation of curve p-256[C]//Workshop on Elliptic Curve Cryptography Standards. Gaithersburg: NIST, 2015: 1-10.

[3] BERNSTEIN D J. Curve25519: New Diffie-Hellman speed records[C]//International Workshop on Public Key Cryptography. Berlin: Springer, 2006: 207-228.

[4] BERNSTEIN D J, DUIF N, LANGE T, et al. High-speed high-security signatures[J]. Journal of Cryptographic Engineering, 2012, 2(2): 77-89.

[5] XIA J, CHENG C N, ZHOU X P, et al. Kunpeng 920: The first 7-nm chiplet-based 64-core ARM SoC for cloud services[J]. IEEE Micro, 2021, 41(5): 67-75.

[6] DONGARRA J. Report on the Fujitsu Fugaku system[R/OL]. (2020-06-22) [2022-11-01]. https://icl. utk. edu/files/publications/2020/icl-utk-1379-2020.pdf.

[7] BERNSTEIN D J, SCHWABE P. NEON crypto[C]//International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2012: 320-339.

[8] FAZ-HERNÁNDEZ A, LÓPEZ J, DAHAB R. High-performance implementation of elliptic curve cryptography us-

ing vector instructions[J]. ACM Transactions on Mathematical Software, 2019, 45(3): 1-35.

[9] KOC C K, ACAR T, KALISKI B S. Analyzing and comparing Montgomery multiplication algorithms[J]. IEEE Micro, 1996, 16(3): 26-33.

[10] MAI L, YAN Y, JIA S L, et al. Accelerating SM2 digital signature algorithm using modern processor features[C]// International Conference on Information and Communications Security. Cham: Springer, 2019: 430-446.

[11] GUERON S, KRASNOV V. Fast prime field elliptic-curve cryptography with 256-bit primes[J]. Journal of Cryptographic Engineering, 2015, 5(2): 141-151.

[12] 国家密码管理局. SM2椭圆曲线公钥密码算法第1部分:总则[S/OL]. (2010-12)[2022-11-10]. https://www.oscca.gov.cn/sca/xxgk/2010-12/17/1002386/files/b791a9f908bb4803875ab6aeeb7b4e03.pdf.

[13] 兰修文. ECC计算算法的优化及其在SM2实现中的运用[D]. 成都:电子科技大学, 2019.
LAN X W. Optimization of ECC Calculation Algorithm and its Application in SM2 Implementation[D]. Chengdu: University of Electronic Science and Technology of China, 2019. (in Chinese)

[14] KNUTH D E. The Art of Computer Programming, Volume 2: Seminumerical Algorithms[M]. Hoboken: Addison-Wesley Professional, 2014.

[15] BRIAN S. The most efficient known addition chains for field element & scalar inversion for the most popular & most unpopular elliptic curves[R/OL]. (2017-05-31) [2022-11-10]. https://briansmith. org/ecc-inversion-addition-chains-01.

[16] Project OpenSSL. bn_mod_exp_mont_consttime subroutine[R/OL]. (2023-06-11) [2023-07-31]. https://github. com/openssl/openssl/blob/master/crypto/bn/bn_exp.c.

[17] Project GmSSL. ecp_sm2z256_mod_inverse subroutine [R/OL]. (2018-09-07) [2022-11-10]. https://github. com/guanzhi/GmSSL/blob/GmSSL-v2/crypto/ec/ecp_sm2z256.c.

[18] ZHOU L, SU C H, HU Z, et al. Lightweight implementations of NIST P-256 and SM2 ECC on 8-bit resource-constraint embedded device[J]. ACM Transactions on Embedded Computing Systems, 2019, 18(3): 1-13.

[19] Project OpenSSL. ecp_nistz256-armv8. pl[R/OL]. (2021-10-01) [2022-11-10]. https://github. com/openssl/openssl/blob/master/crypto/ec/asm/ecp_nistz256-armv8.pl.

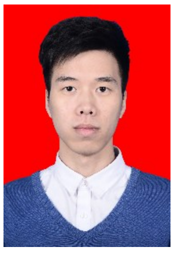
[20] BOOTH A D. A signed binary multiplication technique [J]. The Quarterly Journal of Mechanics and Applied Mathematics, 1951, 4(2): 236-240.

[21] Project OpenSSL. bn_mul_montsubroutine[R/OL]. (2021-10-01) [2022-11-10]. https://github. com/openssl/openssl/blob/master/crypto/bn/asm/armv8-mont.pl.

作者简介

张吉鹏 男,1999年3月出生于山东省济宁市. 现为南京航空航天大学博士生. 研究方向为公钥密码算法、后量子密码算法、密码工程.

E-mail: jp-zhang@outlook.com



黄军浩 男,1995年11月出生于广东省化州市. 现为香港浸会大学、北京师范大学香港浸会大学联合国际学院博士生. 研究方向为公钥密码算法、后量子密码算法、密码工程.

E-mail: huangjunhao@uic.edu.cn

刘 哲(通讯作者) 男,1986年12月出生于山东省济宁市,国家海外高层次人才,现任之江实验室基础理论研究院副院长. 曾获得教育部高校计算机专业优秀教师奖、《麻省理工科技评论》中国区"35岁以下科技创新35人"、阿里巴巴达摩院青橙奖、中国密码学会密码创新奖一等奖等荣誉. 长期从事信息安全领域的研究,在IACR CHES、ACM CCS、IEEE S&P等顶级安全会议和IEEE TC、IEEE TDSC、IEEE TIFS等顶级安全期刊发表学术论文160多篇.

E-mail: zhe. liu@zhejianglab.com