面向GmSSL密码库的SM2算法快速优化实现

乔 晗[1),2)] 王 安[3)] 王 博[2)] 苏长山[2)] 李 根[2)]
唐遇星[2)] 祝烈煌[3)]

[1)](北京理工大学计算机学院 北京 100081)
[2)](飞腾信息技术有限公司 北京 100083)
[3)](北京理工大学网络空间安全学院 北京 100081)

摘 要 GmSSL是由国内密码学专家团队研发的支持国密算法的开源密码库,相比国际主流密码库,不仅严格遵循国家密码标准,还能满足特定的安全需求。SM2算法是GmSSL的重要组成部分,作为国密椭圆曲线密码算法,它在相同安全强度下所需的密钥空间更小、计算效率更高,更适用于资源受限的设备,在国内网络安全领域具有重要意义。尽管GmSSL中SM2算法在功能上相对完善,但在实现细节、性能优化及硬件资源利用方面仍存在明显不足。本文提出了一种系统化的优化方案,针对GmSSL 3中的SM2算法从底层到顶层进行了全方位的优化,在确保密码算法安全性的同时显著提升了算法的实现速度。首先,本文从提升计算效率的角度出发,通过优化模约减算法结构以减少变量间冗余计算,大幅提升了模乘和模平方运算的速度。其次,通过引入并行计算结构,充分利用硬件的并行处理能力,显著提高了点运算的效率。最后,通过采用更高效的算法展开形式,改进了标量乘法的实现方案,进一步提升了标量乘算法实现速度。结合上述三种优化方法,本文对原始GmSSL中的标量乘和签名算法进行了优化实现,优化后的速度分别提升了118.3%和89.3%。此外,与国际主流密码库OpenSSL相比,本文实现的标量乘和签名算法速度分别提升了101.4%和55.8%,这一结果不仅验证了本文优化方案的有效性,也凸显了GmSSL在国际竞争中的潜力。本文的优化方案不仅显著提升了国密算法在实际应用中的性能,对于物联网和移动支付等领域具有重要意义,同时也为后续密码算法研究提供了新的视角和思路。

关键词 GmSSL;SM2;椭圆曲线加密算法;快速实现;标量乘
中图法分类号 TP309 **DOI号** 10.11897/SP.J.1016.2025.00463

Optimized Implementation of the SM2 Algorithm on the GmSSL Cryptographic Library

QIAO Han[1),2)] WANG An[3)] WANG Bo[2)] SU Chang-Shan[2)] LI Gen[2)]
TANG Yu-Xing[2)] ZHU Lie-Huang[3)]

[1)](School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081)
[2)](Phytium Technology Co., Ltd., Beijing 100083)
[3)](School of Cyber Science and Technology, Beijing Institute of Technology, Beijing 100081)

Abstract GmSSL is an open-source cryptographic library developed by a team of Chinese cryptography experts that supports Chinese cryptographic algorithms. Compared to mainstream international crypto-graphic libraries, it not only strictly adheres to national cryptographic standards but also meets specific security requirements. The SM2 algorithm, a crucial component of GmSSL and a state-secret elliptic curvecryptographic algorithm, requires less key space and

收稿日期:2024-01-25;在线发布日期:2024-09-29。**乔 晗**,硕士研究生,主要研究领域为侧信道分析与防护技术。E-mail:3220220955@bit.edu.cn。**王 安**,博士,研究员,主要研究领域为侧信道分析与防护技术。**王 博**(通信作者),博士,副研究员,主要研究领域为硬件安全与处理器安全、软件定义芯片。E-mail:wangbo@phytium.com.cn。**苏长山**,学士,助理研究员,主要研究领域为密码算法及其软硬件安全实现。**李 根**,博士,研究员,主要研究领域为计算机体系结构与微处理器设计。**唐遇星**,博士,研究员,主要研究领域为计算机体系结构与微处理器设计。**祝烈煌**,博士,教授,中国计算机学会(CCF)会员,主要研究领域为密码学、网络与信息安全。

offers higher computational efficiency at the same level of security, making it particularly suitable for resource-constrained devices. It plays a significant role in China′s cybersecurity domain. Despite the relatively comprehensive functionality of the SM2 algorithm in GmSSL, there are notable deficiencies in implementation details, performance optimization, and hardware resource utilization. This paper proposes a systematic optimization approach, enhancing the SM2 algorithm in GmSSL 3 from the bottom up, significantly improving the implementation speed while ensuring the security of the cryptographic algorithms. Initially, this study focuses on enhancing computational efficiency by optimizing the structure of the modular reduction algorithm to minimize redundant calculations between variables, significantly increasing the speed of modular multiplication and squaring operations. Furthermore, introducing parallel computing structures fully utilizes the hardware′s parallel processing capabilities, substantially improving the efficiency of point operations. Finally, adopting a more efficient algorithmic expansion form improves the implementation scheme of scalar multiplication, further accelerating the speed of scalar multiplication implementation. Combining these three optimization methods, this paper optimizes the original scalar multiplication and signing algorithms in GmSSL, with speeds increased by 118.3% and 89.3%, respectively. Moreover, compared to the mainstream international cryptographic library OpenSSL, the speeds of the implemented scalar multiplication and signing algorithms increased by 101.4% and 55.8%, respectively. These results validate the effectiveness of the optimization scheme presented in this paper and highlight the potential of GmSSL in international competition. The optimization scheme significantly enhances the performance of state-secret algorithms in practical applications. It has important implications for fields like the Internet of Things and mobile payments while providing new perspectives and ideas for subsequent cryptographic algorithm research.

Keywords GmSSL; SM2; elliptic curve cryptography; fast implementation; scalar multiplication

1 引 言

随着物联网和数字化时代的到来，信息安全的重要性日益凸显。不同用户设备之间的广泛互联和数据交换，使得数据保护变得尤为关键。在此背景下，对高效且可靠的密码算法需求显著增加，这些算法在保护敏感数据和通信中起着至关重要的作用。与此同时，随着技术的发展和应用的扩展，密码算法的优化实现成为了一个重要议题。这不仅使得密码算法能够支持更大规模的应用，还能更好地满足不断增长的安全需求。因此，对于密码算法的优化实现不仅是技术上的挑战，也是确保信息安全的重要途径。GmSSL[1]是由北京大学关志教授领导的团队开发的一款国产商用密码开源库。与其前身GmSSL 2相比，最新版本GmSSL 3在功能上实现了显著的跨越。它全面覆盖了国密算法、标准和安全通信协议，支持包括移动端在内的主流操作系统和处理器。此外，GmSSL 3支持国产密码硬件，如密码钥匙和密码卡，并提供功能丰富的命令行工具及多种编程语言的编程接口。另外，GmSSL 3兼容国际主流密码库OpenSSL[2]的API，这为用户在原有基于OpenSSL的应用场景中快速切换到国密算法提供了便利。与OpenSSL相比，GmSSL 3严格遵循中国国家密码标准和规范，因此，在国内环境下，使用GmSSL 3不仅更符合国家规定，同时也能满足特定的安全需求。

尽管GmSSL 3作为一个密码库在功能性方面具备一定优势，但其在实现速度上相较于OpenSSL或其他密码库仍存在一些劣势。主要表现在以下三个方面：

(1) 优化不足：OpenSSL等国际密码库受益于全球开发者社区的广泛贡献和持续优化，而GmSSL 3的开发与优化主要依赖于国内开发者。这可能导致GmSSL 3在某些优化方面无法与国际社区保持同步。

(2)算法实现差异：不同密码库采用不同的算法实现方式，这可能导致效率差异。GmSSL 3中某

些密码算法的实现效率可能低于其他密码库。

（3）硬件利用不足：相较于国际上的其他密码库，GmSSL 3在利用硬件特性（如并行处理能力、处理器的密码指令扩展等）进行优化方面存在不足，这限制了其在执行密码算法时的性能提升。

SM2算法是国家密码管理局于2010年发布的一种基于椭圆曲线密码学（Elliptic Curve Cryptography，ECC）的国产公钥密码算法[3]，SM2的设计考虑了国内的密码需求以及安全标准，因此其在国内的网络安全领域具有重要的战略意义。SM2算法包括数字签名、密钥交换和公钥加密三部分，广泛应用于金融、政务、电信等关键信息基础设施领域。如图1所示，从整体架构上来讲，SM2算法可以分为四个层次。域运算层：椭圆曲线密码学中的基本运算，包括模加、模减、模乘、模逆等运算，是SM2算法的基础。点运算层：椭圆曲线上的点加、倍点等运算，依赖于域运算层。多倍点运算层：基于点运算层的标量乘算法，是SM2算法的核心运算之一。SM2协议层：包括数字签名、密钥交换和公钥加密等具体的密码协议，依赖于多倍点运算层。

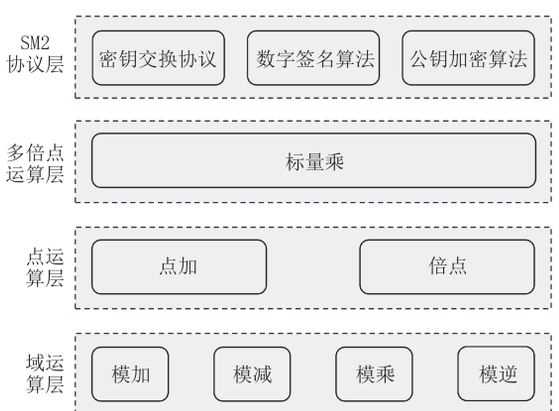

图1　SM2算法实现结构

在数据安全的需求日益增加的今天，快速的加密和解密速度是保障用户体验和处理大量数据的关键。尤其是在金融服务、云计算和大规模物联网设备中，速度的提升不仅能显著减少系统的延迟，增加事务处理的吞吐量，还能优化能源使用和减少运行成本。因此，提升GmSSL 3的实现速度，不仅是提升密码库竞争力的必要条件，也是满足现代安全需求的实际需要。

以往的研究中，大多数论文通常专注于密码算法的特定层面进行优化。举例来说，在域运算层面，研究主要集中于优化耗时较多的操作，如模乘和模逆运算。Karatsuba采用分而治之的思想，将大整数乘法问题分解为更小的子问题，并利用乘法的分配律和恒等式来减少乘法操作的次数，将传统的大整数乘法时间复杂由$O(n^2)$降低到$O(n^{1.585})$[4]。Montgomery将大数转化到Montgomery域上，从而将大数模乘中耗时较大的乘法操作转变为移位和求余运算，实现了指数级别的运算加速[5]。Solinas引入了广义梅森素数，这些素数允许更高效的模运算。利用这些特殊素数，可以显著减少模乘和模逆运算的复杂度。Solinas的方法在素数域运算中得到了广泛应用，特别是在需要高效计算的大整数运算中[6]。Hankerson等人通过坐标系的映射变换，将仿射坐标系下的点转换到其他坐标系上从而避免了模逆操作[7]。Wang等人基于单指令流多数据流（Single Instruction Multiple Data，SIMD）协处理器提出了针对固定素数模p的向量模乘运算以减少乘法运算的中间结果，最终将NISTP192、Secp256k1和Numsp256d1三种曲线上的模乘运算分别降低到205、310和306个时钟周期，与乘数约减算法[8]相比速度提高了32%[9]。在点运算层面，Miller提出了利用点斜率的快速点加运算，这种方法显著减少了点加运算的复杂度[10]。Márquez等人使用NEON指令集并行实现了模乘运算，进而将点加和倍点运算重新编排以减少模乘运算的运行时间，最终在三种不同的ECC曲线上将标量乘算法的性能提升了32%～38%[11]。Van等人提出了改进的点算术运算算法，通过将乘法对和平方对组合来减少中间计算时间，在ARMv8架构下与顺序实现点运算的速度相比提升了10%～20%[12]。在标量乘层面，Montgomery提出使用Montgomery阶梯算法处理标量乘算法，使得每一次迭代的计算不依赖于输入数据的位值[13]，从而抵抗了侧信道攻击[14]。Cohen等人提出了GLV算法，这种算法通过利用椭圆曲线的特殊结构，将标量乘法分解为两个较小的乘法，从而提高了运算速度。该方法在特定类型的椭圆曲线（如GLV曲线）上特别有效，显著减少了计算时间[15]。Mahdavi等人将混合坐标策略和直接计算应用于不同的标量乘算法，找到了混合坐标策略和直接计算于不同标量乘算法实现的最佳组合，从而使计算成本和内存消耗达到最优[16]。Zhao等人提出全精度乘法器从而优化了标量乘算法，在硬件评估阶段，他们的SM2架构每秒可以执行超过49 000次

的标量乘算法[17]。Huang等人利用AVX2实现的基于Co-Z运算的Montgomery阶梯算法的运行效率与原实现相比提升了1.31倍[18]。

本文对GmSSL 3(以下称为GmSSL)密码库中的SM2算法进行快速优化实现,针对域运算层、点运算层以及多倍点运算层实施了一系列改进和优化措施。这些改进在不影响GmSSL的优势特性的前提下,显著提高了SM2算法实现速度。本文的创新点主要体现在以下三个方面:

(1) 在域运算层,本文首次提出了一种针对SM2快速模约减过程的优化算法,具体而言,算法通过减少变量的重复计算以及后续对模数p的减法操作,大大提高了模乘和模平方运算的实现速度。

(2)在点运算层,本文首次在ARM架构下使用NEON指令集针对模乘和模平方运算进行并行实现,以此优化点加和倍点运算,有效提升了点运算的实现速度。

(3)在标量乘运算层,本文引入了非相邻表示型(Non Adjacent Form, NAF)展开法[19]取代传统的二进制展开法。与原始二进制展开标量乘算法相比,NAF展开算法减少了标量乘算法中点加运算的数量,从而提高了标量乘算法的实现速度。

综合以上三种不同的优化实现方法,本文相较于原始GmSSL中的标量乘法算法,速度提升了118.3%。值得一提的是,这一结果不仅证明了本文优化策略在提升SM2算法性能方面的有效性,还为学术界及产业界提供了一个系统性的优化方案。无论是在GmSSL还是其他密码库中,本文的优化方案都有望带来类似的性能提升效果。对于致力于优化其他密码库中ECC算法的研究者,本文提出的优化思路和方法同样具有重要的参考价值。本文所有的贡献都集成到了GmSSL库,读者可以通过以下链接访问:https://github.com/sususu98/GmSSL/。

2 预备知识

2.1 NEON指令集

NEON是ARM处理器架构中的一种指令集扩展,用于加速多媒体、图像处理和信号处理等应用。NEON指令集采用了SIMD技术,可以同时对多个数据元素执行相同的操作,从而提高数据处理的效率和性能。在ARMv8架构中,NEON指令集提供了32个128位的向量寄存器($V_0 \sim V_{31}$),同时,这些寄存器又可进一步拆分为32个64位D寄存器($D_0 \sim D_{31}$)或32个32位的S寄存器($S_0 \sim S_{31}$),以此提供了强大的并行计算和向量处理能力,如图2所示。而对于并行运算的数据对,为了确保并行运算的准确性和效率,参与并行计算的数据对必须具有相同的比特位数。该指令集支持的向量元素大小为8位、16位、32位以及64位,对应支持16路、8路、4路及2路并行SIMD操作。例如汇编指令:

UMULL V0.2D, D1.2S, D2.2S

表示对存储在64位D_1寄存器中一对32位向量元素与存储在64位D_2寄存器中一对32位数据做乘法,得到一对64位数据,并将结果储存在128位V_0寄存器中。

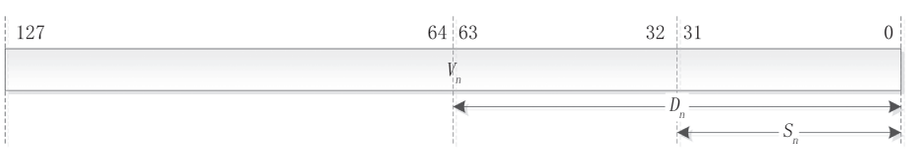

图2 AArch64 NEON寄存器

AVX指令集也是一种SIMD指令集,它是x86处理器架构中的一种指令集扩展,用于提高浮点运算和整数运算的性能。AVX2是AVX指令集的扩展版本,进一步增加了整数运算和位操作指令,使其在整数运算和浮点运算方面都有显著提升。AVX2引入了256位的整数向量操作,使得每个寄存器可以存储8个32位整数或4个64位整数。但AVX主要用于x86架构的处理器,而NEON指令集主要用于ARM架构的处理器。两者在各自的硬件平台上都发挥着重要的作用,为提升计算性能提供了有效的手段。

2.2 雅可比坐标系

SM2算法最底层的域运算在固定素数域F_p上进行,其中素数p定义为$p=2^{256}-2^{224}-2^{96}+2^{64}-1$[3]。域运算为点运算提供基础,后者定义在椭圆曲线$E(F_p)$上,其中$E(F_p)$表示在素数域F_p上定义的椭圆曲线点的集合。在仿射坐标系下,椭圆曲线上的点集记为$E(F_p)=\{(x,y)|xy\in F_p$且满足曲线方

程$y^2=x^3+ax+b\}\cup\{O\}$，其中O是椭圆曲线的无穷远点。虽然仿射坐标系下的椭圆曲线表达形式简洁且直观，但在执行点加和倍点运算时，会涉及到复杂且计算成本高昂的除法运算[7]，这对算法性能构成了显著的制约。因而在实现点运算时，通常需要将运算点的仿射坐标转换到其他坐标系下，以避免除法运算所带来的时间消耗，在GmSSL中SM2算法的点加和倍点运算是将仿射坐标转换到雅可比加重射影坐标系下，在仿射坐标系下假设点$P_1=(x_1,y_1)\in E(F_p)\backslash\{O\}$，$P_2=(x_2,y_2)\in E(F_p)\backslash\{O\}$则$P_3=(x_3,y_3)=P_1+P_2$的仿射坐标可由公式(1)给出：

$$\begin{aligned}x_3&=\left(\frac{y_2-y_1}{x_2-x_1}\right)^2-x_1-x_2\\y_3&=\left(\frac{y_2-y_1}{x_2-x_1}\right)(x_1-x_3)-y_1\end{aligned}\tag{1}$$

同理$P_3=(x_3,y_3)=2P_1$的仿射坐标可由公式(2)给出：

$$\begin{aligned}x_3&=\left(\frac{3x_1^2+a}{2y_1}\right)^2-2x_1\\y_3&=\left(\frac{3x_1^2+a}{2y_1}\right)(x_1-x_3)-y_1\end{aligned}\tag{2}$$

下面以I、M、S分别表示在素数域F_p上的逆元计算、乘法以及平方运算，则在仿射坐标系下，如若不考虑域上耗时很小的加法和减法运算，可以估算得到点加和倍点运算的计算成本分别为：1I+2M+1S和1I+2M+2S，其中除法运算往往要比模乘运算慢很多(1I≈100M)[7]。而通过映射：

$$x=\frac{X}{Z^2},y=\frac{Y}{Z^3}\tag{3}$$

即可将雅可比加重射影坐标系中的点转换到仿射坐标系下，在雅可比坐标系下，模逆在迭代过程中被省略，点加和倍点运算的计算成本分别降低到：8M+3S和4M+4S，具体的计算成本对比如表1所示：

表1　不同坐标系下的点运算计算成本对比

坐标系	点加运算	倍点运算
仿射坐标系	1I+2M+1S	1I+2M+2S
雅可比坐标系	8M+3S	4M+4S

2.3　模乘和模平方

在域运算层面，模乘和模平方运算主要分为两个阶段：大数乘法运算和模约减运算。在GmSSL实现中，大数乘法运算采用了操作数扫描法。具体而言，对于两个以2^{32}为基底大整数$A,B\in[0,p]$，其中$A=(A_7,\cdots,A_1,A_0)$，$B=(B_7,\cdots,B_1,B_0)$，首先通过外层循环扫描操作数A_i，其次内层循环则遍历操作数B_j，最后将两个操作数乘积的高位作为进位参与下一轮内循环，低位作为累加值参与下一轮外循环。

对于大数乘法之后的模约减运算，即将A,B的乘积$C\in[0,p^2]$对固定的素数p进行取模运算，以获得乘积C在素数域F_p上的模约减结果。在SM2算法中使用的素数p是一种广义梅森素数，其特殊的形式使我们能够采用特定的约减算法来降低计算复杂度。广义梅森素数定义为形如$f(2^n)$的素数，其中$f(x)$为低次多项式。因此广义梅森素数可以表示成少量2的幂次多项式的和或差，此外SM2算法中素数多项式$f(2^n)$中多项式的幂次均为32的倍数，基于这一特性，可以将待约减中间值按照32位分割，从而大大提高了64位机器上模约减算法的计算速度。在GmSSL中，SM2算法在素数域F_p上的模约减运算是通过现有的快速归约算法[17]来实现的，该算法的具体实现细节如算法1所示。

算法1. 素数域F_p上的快速模约减算法

输入：以2^{32}为基底的大数$C=(c_{15},\cdots,c_1,c_0)$；其中$0\leqslant C<p^2$

输出：$C\ mod\ p$

阶段1：预计算大数Z

1. $s_1=(c_7,c_6,c_5,c_4,c_3,c_2,c_1,c_0)$
2. $s_2=(c_{15},c_{14},c_{13},c_{12},c_{11},0,c_9,c_8)$
3. $s_3=(c_{14},0,c_{15},c_{14},c_{13},0,c_{14},c_{13})$
4. $s_4=(c_{13},0,0,0,0,0,c_{15},c_{14})$
5. $s_5=(c_{12},0,0,0,0,0,0,c_{15})$
6. $s_6=(c_{11},c_{11},c_{10},c_{15},c_{14},0,c_{13},c_{12})$
7. $s_7=(c_{10},c_{15},c_{14},c_{13},c_{12},0,c_{11},c_{10})$
8. $s_8=(c_9,0,0,c_9,c_8,0,c_{10},c_9)$
9. $s_9=(c_8,0,0,0,c_{15},0,c_{12},c_{11})$
10. $s_{10}=(c_{15},0,0,0,0,0,0,0)$
11. $s_{11}=(0,0,0,0,0,c_{14},0,0)$
12. $s_{12}=(0,0,0,0,0,c_{13},0,0)$
13. $s_{13}=(0,0,0,0,0,c_9,0,0)$
14. $s_{14}=(0,0,0,0,0,c_8,0,0)$
15. $Z=s_1+s_2+2s_3+2s_4+2s_5+s_6+s_7+s_8+s_9+2s_{10}-s_{11}-s_{12}-s_{13}-s_{14}$

阶段2：计算$Z\bmod p$

16. while $Z>p$ **do**

17. $Z \leftarrow Z-p$
18. end while
19. return Z

算法1首先预存了14个256位的整数$s_1 \sim s_{14}$，这些预存整数为后续的计算节省了时间。随后，通过执行14次加法操作和4次减法操作，便可得到一个与乘积结果C同余的整数Z，其中$Z \in [0, 14p]$，由于算法对256位的整数共执行了14次的加法操作，因此最终计算的结果Z的长度至多为270位。这会大大减少了后续对p取模运算的计算成本。最终，通过若干次减法操作，便可高效地得到最终约减结果。

2.4 二进制展开

标量乘算法是整个ECC算法中耗时最大的部分，它决定着椭圆曲线密码体制的运算速度，一般将标量乘算法定义为$Q=k \cdot P$，其中Q、P是定义在域F_p上的椭圆曲线E上的点，k是一个整数，而在已知Q和P的情况下基本不可能将标量k得到，这个难解的问题，也叫椭圆曲线离散对数问题，所以标量乘算法也是ECC算法安全性的基础。

常规标量乘算法是通过二进制展开将乘法拆解为若干个点加和倍点运算。即对正整数k进行二进制展开，从高位到低位依次遍历每一位k_i。对于每一位，算法首先执行一个倍点运算，然后根据k_i的值（即若k_i为1）有条件地执行一个点加运算。这一系列操作的具体实现细节如算法2所示。

算法2. 基于二进制展开的标量乘算法

输入：标量$k=(k_{l-1}, \cdots, k_1, k_0)_2$，椭圆曲线点$P \in E(\mathrm{F_p})$

输出：$Q=k \cdot P$

1. $Q \leftarrow O$
2. for $i=l-1$ to 0 do
3. $Q \leftarrow 2Q$
4. if $k_i == 1$ then $Q \leftarrow Q+P$
5. end for
6. return Q

对于算法2，假设标量k二进制展开后的长度为l_{bin}，比特位“1”和“0”出现的概率相同，所以标量k经过二进制展开后的平均汉明重量为

$$\mathrm{HW}_{\mathrm{bin}}(k)=\frac{l_{\mathrm{bin}}}{2} \tag{4}$$

我们分别以A和D表示点加和倍点运算，那么二进制展开算法的平均计算成本为

$$\left(\frac{l_{\mathrm{bin}}}{2}\right)\mathrm{A}+l_{\mathrm{bin}}\mathrm{D} \tag{5}$$

3 SM2算法优化方案

3.1 基于快速模约减的域运算优化设计

原始GmSSL库的模约减算法，即算法1通过一系列的加法和减法操作将乘积$C \in [0, p^2]$简化为一个比特位长度不超过270位的大数$Z \in [0, 14p]$，然后通过若干次减法操作得到最终的模约减结果。此预处理步骤显著降低了直接进行大数取模运算的时间开销。本文提出了一种针对SM2快速模约减过程的优化算法，通过减少变量的重复计算以及后续对模数p的减法操作，提高模乘和模平方的实现速度。

首先，在执行新定义整数$s_1 \sim s_{14}$的加减运算时，会从$s_1 \sim s_{14}$的最低32位开始相加，然后考虑进位向上继续计算，直到最高32位。整个算法中会有多个相同的加减运算操作反复出现，如$c_8+c_9+c_{10}+c_{11}+c_{12}$在整个过程中会出现3次。通过将这些多次执行的运算结果预先使用中间值进行存储，可以减少重复的计算，从而大幅度减少计算大数Z的时间，具体实施见算法3。

算法3. 在素数域上改进的大数加法

输入：以2^{32}为基底的大数$C=(c_{15}, \cdots, c_1, c_0)$；其中$0 \leqslant C < p^2$

输出：$Z=(carry, z_7, \cdots, z_1, z_0)$

阶段1：预存变量以优化后续操作

1. $r_0 \leftarrow c_8+c_9+c_{10}+c_{11}$, $r_1 \leftarrow c_8+c_{13}$, $r_2 \leftarrow c_9+c_{14}$
2. $r_3 \leftarrow c_{14}+c_{15}$, $r_4 \leftarrow c_{13}+r_3$, $r_5 \leftarrow c_{12}+r_4$, $r_0 \leftarrow r_0+r_5$

阶段2：按列进位计算$Z=(carry, z_7, \cdots, z_1, z_0)$

3. $(t, z_0) \leftarrow c_0+r_0+r_4$
4. $(t, z_1) \leftarrow t+c_1+r_0+r_4-r_1$
5. $(t, z_2) \leftarrow t+c_2+0\mathrm{x}400000000-r_1-r_2$
6. $(t, z_3) \leftarrow t+c_3+r_5+r_1+c_{11}+0\mathrm{xFFFFFFFC}$
7. $(t, z_4) \leftarrow t+c_4+r_5+r_2-1$
8. $(t, z_5) \leftarrow t+c_5+r_4+c_{10}+c_{15}$
9. $(t, z_6) \leftarrow t+c_6+c_{11}+r_3$
10. $(carry, z_7) \leftarrow t+c_7+r_0+r_5+c_{15}$
11. return Z

算法3针对算法1中阶段1进行优化，其包括两个关键阶段。在第一阶段，算法预存了若干需要重复计算的中间变量，以优化整个运算过程的效率。在第二阶段，算法对整数$s_1 \sim s_{14}$从低位到高位逐步执行加减运算。每次运算保留低32位结果z_i作为本轮的输出，而超出的比特位作为进位值$carry$传递至下一轮运算。此过程逐步构建出中间变量Z。

算法3的输出值与算法1中阶段1的输出值相同，均为一个不超过270位的整数$Z\in[0,14p]$，而我们需要的是在区间$[0,p)$上的模约减结果，因此算法1中阶段2在最坏情况下还需要执行13次减法操作。

然而Z的进位值$carry$实际上是区间[0，13]上的整数，我们可以利用这个已知的进位值以及固定的模数p来确定需要约减的最大整数，即$(carry+1)\cdot p$。之后仅需要执行$Z-(carry+1)\cdot p$次减法操作，根据在减法过程中是否存在借位，有条件执行一次加法操作，即可得到最终$c \bmod p$的约减结果，具体流程见算法4。

算法4. 在素数域F_p上的改进的mod p算法

输入：以2^{32}为基底的大数$Z=(carry, z_7, \cdots, z_1, z_0)$

输出：$Z \bmod p$

1. $n \leftarrow carry+1$

阶段1：预计算$n\cdot p=(p_4, p_3, p_2, p_1, p_0)$

2. $p_4 \leftarrow n-1$
3. $p_3 \leftarrow !(n \ll 32)$
4. $p_2 \leftarrow$ 0xFFFFFFFF
5. $p_1 \leftarrow (n-1)-(n \ll 32)$
6. $p_0 \leftarrow -n$

阶段2：计算$Z \bmod p$

7. $Z \leftarrow Z-n\cdot p$
8. if $Z<0$ then $Z \leftarrow Z+p$
9. return Z

在域运算层面，我们针对模乘和模平方的模约减阶段进行优化. 具体而言，我们针对算法1的两个阶段进行了不同程度的优化实现。对于阶段1，我们注意到在执行加减运算时会重复计算相同的表达式通过预先存储这些多次执行的运算结果，我们可以避免重复计算，从而显著减少计算大数Z所需的时间。这种优化方法通过引入中间值存储的方式，有效地提高了算法的效率。

对于阶段2，我们注意到算法3输出结果的进位值$carry$已知且固定，我们可以利用这个已知的进位值和模数p来确定需要约减的最大整数，结果只需一次模减运算以及至多一次模加运算，即可得到最终的$c \bmod p$的约减结果。这种优化方法减少了不必要的减法操作，提高了阶段2的执行效率。

3.2　基于NEON并行计算的点运算优化设计

原始GmSSL库中的点加和倍点运算，通过坐标系的变换避免了耗时较大的除法运算，从而将点加和倍点运算的计算成本分别降低到8M+3S和4M+4S。尽管如此，现有的模乘和模平方运算仍以顺序方式执行。在点加和倍点运算的执行过程中，大部分素数域运算操作实际上并不具有数据依赖性，这为并行化提供了可能。本文在ARM架构下使用NEON指令集针对SM2算法中的模乘和模平方运算进行并行实现，以此提升点运算的实现速度。由于GmSSL中SM2算法的数据存储结构刚好与NEON寄存器相适配，因此使用NEON指令集并行处理模乘或模平方运算会有较大的速度提升，基于此本文对这些非数据依赖性的模乘和模平方运算进行了并行优化处理。

以素数域F_p上的模乘运算为例，我们通过NEON指令集并行地实现了两个256位大数的乘法运算。鉴于乘法操作可能产生比特位溢出，以及NEON的向量寄存器最大仅支持并行处理64位向量，因此我们仅能并行执行32位的大数乘法。因而对于256位的大数乘法，这要求将其拆解为8×32位的向量进行计算。并行256位大数乘法的具体过程如算法5所示。该算法以两对8×32位向量作为输入，以一对16×32位向量作为输出。

算法5. 素数域F_p上的256位的并行大数乘法

输入：$A=(A_7, \cdots, A_1, A_0)$，$B=(B_7, \cdots, B_1, B_0)$；
$C=(C_7, \cdots, C_1, C_0)$，$D=(D_7, \cdots, D_1, D_0)$

输出：$M=(M_{15}, \cdots, M_1, M_0)=A\cdot B$；
$N=(N_{15}, \ldots, N_1, N_0)=C\cdot D$

1. $M=0, N=0$
2. for $i=7$ to 0 do
3. $H_1=0, H_2=0$
4. for $j=7$ to 0 do
5. $(H_1, L_1)=M_{i+j}+A_i\cdot B_j+H_1$
6. $(H_2, L_2)=N_{i+j}+C_i\cdot D_j+H_2$
7. $M_{i+j}=L_1, N_{i+j}=L_2$
8. end for
9. $M_{i+8}=H_1, N_{i+8}=H_2$
10. end for
11. return M, N

在实现256位大数并行乘法时，受限于NEON指令集寄存器的能力，只能处理一对32位向量的乘法运算。因此，算法的输入和输出均是以232为底的向量。通过16个循环迭代，算法完成所需的并行乘法运算。图3展示了第$i+j$个循环中NEON寄存器并行数据处理的过程，其中H和L分别表示运算结果的高32位和低32位。

在点加和倍点运算中，本文采用了NEON指令集以并行方式执行两个无数据依赖的模乘或模平方

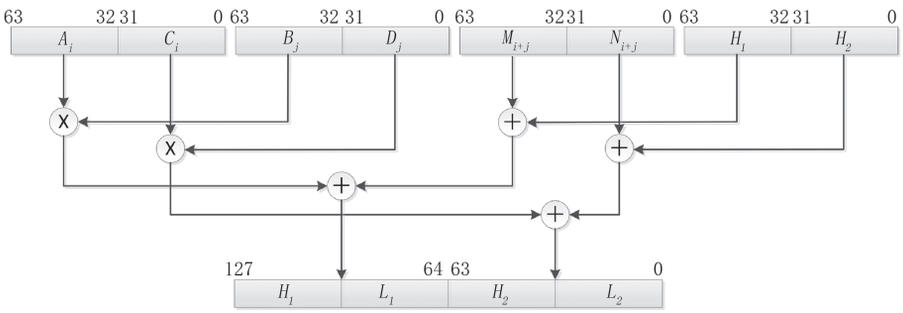

图3 第$i+j$个循环的运算结果

运算，传统的点加和倍点运算并没有针对并行处理性能进行优化，鉴于此，本研究基于NEON指令集的特性，重新设计了点加和倍点运算的实现方式，以实现模乘和模平方运算的无数据依赖并行执行。这种优化策略有效降低了按顺序执行模乘和模平方运算所引起的时间开销。具体优化后的点加和倍点运算实现如算法6和算法7所示。其中，{NEON}表示该行操作使用NEON指令集并行实现，{C}表示该行操作使用C语言顺序实现。

算法6. 基于NEON并行实现点加运算

输入：雅可比坐标系下的点$P_1=(X_1, Y_1, Z_1)$，仿射坐标系下的点$P_2=(x_2, y_2)$

输出：$P_3=P_1+P_2=(X_3, Y_3, Z_3)$

1. $T_1=Z_1^2$ {C}
2. $T_2=T_1\cdot Z_1,\ T_5=T_1\cdot x_2$ {NEON}
3. $T_1=T_5-X_1$ {C}
4. $T_2=T_2\cdot y_2,\ Z_3=Z_1\cdot T_1$ {NEON}
5. $T_2=T_2-Y_1$ {C}
6. $T_3=T_1^2,\ X_3=T_2^2$ {NEON}
7. $T_4=T_3\cdot T_1,\ T_6=T_3\cdot X_1$ {NEON}
8. $T_1=2T_6$ {C}
9. $X_3=X_3-T_1$ {C}
10. $X_3=X_3-T_4$ {C}
11. $T_3=T_6-X_3$ {C}
12. $T_3=T_3\cdot T_2,\ T_4=T_4\cdot Y_1$ {NEON}
13. $Y_3=T_3-T_4$ {C}
14. return(X_3, Y_3, Z_3)

在GmSSL中，模乘和模平方运算的实现是相同的。这一特点使得我们在进行并行实现时，可以将相互独立且无数据依赖的模乘和模平方运算结合起来执行。为了描述这种优化实现的计算成本，我们引入M′作为双重乘法的计算成本标记。原始顺序实现算法与优化后的并行实现算法在计算成本上的对比如表2所示。

算法7. 基于NEON并行实现倍点运算

输入：雅可比坐标系下的点$P_1=(X_1, Y_1, Z_1)$

表2 顺序与并行点运算计算成本对比

实现方法	点加运算	倍点运算
顺序实现	8M+3S	4M+4S
并行实现	5M′+1S	4M′

输出：$P_3=2P_1=(X_3, Y_3, Z_3)$

1. $Y_3=2Y_1$ {C}
2. $T_1=Z_1^2, T_4=Y_3^2$ {NEON}
3. $T_2=X_1-T_1$ {C}
4. $T_2=T_2\cdot T_1,\ Z_3=Y_3\cdot Z_1$ {NEON}
5. $T_2=3T_2$ {C}
6. $T_3=T_4\cdot X_1,\ X_3=T_2^2$ {NEON}
7. $T_1=2T_3$ {C}
8. $X_3=X_3-T_1$ {C}
9. $T_1=T_3-X_3$ {C}
10. $T_1=T_1\cdot T_2,\ Y_3=T_4^2$ {NEON}
11. $Y_3=Y_3/2$ {C}
12. $Y_3=T_1-Y_3$ {C}
13. return(X_3, Y_3, Z_3)

3.3 基于NAF展开的标量乘运算优化设计

原始GmSSL库中通过二进制展开实现标量乘算法，对于标量k的二进制展开式，比特“1”相比于比特“0”需要多执行一个点加运算，因而可以通过重构标量k的展开式来减少比特位非零出现的次数，进而减少点加运算，提高标量乘算法的实现效率。

本文引入了NAF展开算法，它是一种有符号的非相邻型展开算法。以标量k的NAF表示为例：

$$k=\sum_{j=0}^{l_{\mathrm{NAF}}-1} k_j 2^j, k_j\in\{0, 1, -1\} \tag{6}$$

这种展开方式可以避免连续出现两个非零值，从而降低非零k_j出现的频率，具体的算法流程见算法8。

算法8. 基于NAF展开的标量乘算法

输入：标量k，椭圆曲线点$P\in E(F_p)$

输出：$Q=k\cdot P$

阶段1：计算标量k的NAF展开形式$NAF(k)=$

$(k_{l-1},\cdots,k_1,k_0)$

1. $i\leftarrow 0$
2. while $k\geqslant 1$ do
3. if k是奇数 then
4. $k_i\leftarrow 2-(k\ mod\ 4)$, $k\leftarrow k-k_i$
5. else $k_i\leftarrow 0$
6. $k\leftarrow k/2, i\leftarrow i+1$
7. end while

阶段2：基于NAF展开式计算标量乘

8. $Q\leftarrow O, l\leftarrow lens(NAF(k))$
9. for $i=l-1$ to 0 do
10. $Q\leftarrow 2Q$
11. if $k_i==1$ then $Q\leftarrow Q+P$
12. if $k_i==-1$ then $Q\leftarrow Q-P$
13. end for
14. return Q

算法8中*lens*()函数是用于计算和返回给定输入值的长度。不妨假设标量k经阶段1展开后的长度为l_{NAF}，并且展开式中“1”，“-1”和“0”出现的概率相同，那么平均汉明重量为：

$$HW_{NAF}(k)=\frac{l_{NAF}}{3} \tag{8}$$

标量k的NAF展开式的长度至多比二进制展开式的长度大1位，即有$l_{NAF}\leqslant l_{bin}+1$，因而NAF算法平均计算成本为：

$$\left(\frac{l_{NAF}}{3}\right)A+l_{NAF}D\approx\left(\frac{l_{bin}}{3}\right)A+l_{bin}D \tag{8}$$

若标量k的长度为256位，则相较于二进制展开算法，NAF标量乘展开算法可以减少大约43个点加运算。

4　实验结果分析

本节内容综合第三节中提出的三种优化方案，对GmSSL3密码库上SM2数字签名算法进行快速优化实现。其中签名生成过程耗时最大的为标量乘算法$k\cdot G$，其性能直接决定了整个算法的效率。而签名验证过程耗时最大的为多标量乘算法$k\cdot U+l\cdot V$，其实际上可以看作两个标量乘算法和一次点加运算的结果，同时点加运算相对于标量乘算法，计算复杂度较低。因此在优化实现的意义上，本文仅对标量乘算法$k\cdot G$以及签名生成算法进行时钟周期的测量和比较。

另外，SM2签名生成算法还涉及其他操作，例如消息哈希、随机数生成、签名值计算等。这些操作在原始的GmSSL库中已经有相对高效且可靠的实现，因而除标量乘算法外的其他部分我们都继续使用了GmSSL库中的原始代码，没有进行额外的改动。下文我们将SM2签名生成算法简记为SM2签名算法。

下面我们将介绍实验环境的配置以及用于评估算法性能的测试方法。此后，我们将对比这些优化方法在速度提升方面的效果，并分析它们相对于原始GmSSL实现的速度提升比率。最后，为了全面评估本文提出的优化实现的效果，我们将其与原始GmSSL中的SM2签名算法以及OpenSSL库中的SM2签名算法进行性能对比分析。

4.1　实验环境配置

我们在Linux系统下，使用时钟频率2.6GHz的飞腾腾锐D2000高性能桌面处理器基于GmSSL3实现了SM2算法，该处理器集成8个FTC663处理器核，每核拥有8MB的L2缓存和4MB的L3缓存，兼容64位ARMV8指令集，本文的优化实现方法适用于所有支持NEON指令集的ARM架构的处理器。为了评估算法的性能，我们通过访问ARM的性能监视器单元（Performance Monitoring Unit，PMU）中的周期计数寄存器（Performance Monitors Cycle Count Register，PMCCNTR_EL0），获得了不同操作准确的执行时间，这种方法使我们能够精确地比较原始密码库中的SM2算法和本文提出的改进后SM2算法之间的性能差异，PMCCNTR_EL0默认只允许内核权限的访问，为了消除权限切换带来的性能差异，我们赋予其用户空间的访问权限，并在密码库的内部插入PMCCNTR_EL0的读取代码以获得准确的性能数据。此外，为了降低因数据输入或环境等不可控因素导致的测量误差，我们对每项操作执行了10 000次重复测试，并取其平均值作为待测算法的执行时间。这种方法确保了测试结果的准确性和可靠性，并且为进一步地分析和优化提供了稳健的数据支撑。

4.2　实验结果与分析

本节内容首先展示了域运算优化、点运算优化以及标量乘运算优化与原始GmSSL库实现SM2算法不同操作所需的时钟周期对比结果，下文我们将原始GmSSL中未经优化的SM2算法称为原始GmSSL。

在域运算层面，我们针对原始GmSSL中的模约减过程进行优化，通过减少变量的重复计算以及

后续对模数p的减法操作，提高模乘和模平方的实现速度。表3展示了在相同硬件和软件测试环境下，原始GmSSL库与依据算法3和算法4改进后的各种运算操作所消耗的时钟周期数。优化后，模乘和模平方运算的时耗从原始的580个时钟周期降低至298个时钟周期，速度提升了94.6%。此外，相较于模乘和模平方运算的速度提升比率，点加、倍点以及标量乘运算的速度仅提升了75.2%至79.6%，这是因为点加和倍点运算实现过程还有一部分未被优化的模加减运算。最终，对于SM2签名算法，域运算优化实现后的算法与原始GmSSL相比速度提升了59.1%，这一改进有望在实际应用中实现更高的能效比。

表3 域运算优化与原始GmSSL库实现的时钟周期对比

运算操作	时钟周期数		提升比率
	GmSSL[1]	域运算优化	
模乘/平方	580	298	94.6%
点加	6938	3864	79.6%
倍点	5488	3133	75.2%
标量乘	2 297 357	1 303 154	76.3%
SM2签名	2 739 607	1 721 517	59.1%

在点运算层面，原始GmSSL中的点运算实现中存在无数据依赖的模乘和模平方运算，我们使用NEON指令集并行实现了这些无数据依赖性的运算操作。为了量化并行实现的效果，我们将顺序实现一个模乘运算与并行实现两个模乘运算所需的时钟周期数进行对比，同时还对比了AVX指令集并行实现提升比率。具体的实验结果如表4所示。由于CPU性能及架构的不同，我们主要关注并行实现对于模乘运算速度的提升比率。对于Curve25519椭圆曲线算法，模乘运算的顺序实现消耗51个时钟周期，而使用AVX并行实现两个模乘运算只消耗64个时钟周期，提升比率为59.4%。对于Curve448椭圆曲线算法，模乘运算的顺序实现消耗110个时钟周期，AVX并行实现消耗130个时钟周期，提升比率为69.2%。对于SM2椭圆曲线算法，模乘运算的顺序实现消耗65个时钟周期，AVX并行实现消耗81个时钟周期，提升比率为60.5%。而本文顺序实现的模乘运算需要消耗528个时钟周期，使用NEON指令集并行实现只需要消耗567个时钟周期，提升比率达到86.2%。尽管NEON指令集的绝对时钟周期数较高，但其并行实现的提升比率显著高于AVX指令集，这可有效证明NEON指令集在实现并行模乘运算时的优越性。

表4 不同并行实现方法在模乘运算中的速度提升对比

椭圆曲线	实现方法	模乘消耗的时钟周期		提升比率
		顺序实现	并行实现	
Curve25519	AVX2[20]	51	64	59.4%
Curve448	AVX2[20]	110	130	69.2%
SM2	AVX2[18]	65	81	60.5%
	NEON	528	567	**86.2%**

使用NEON指令集优化后的点运算所需时钟周期数如表5所示。通过NEON指令集优化后，点加运算从6938个时钟周期降至6502个，速度提升了6.7%；倍点运算从5488个时钟周期降至5030个，速度提升了9.1%；而SM2签名算法的提升幅度最小，仅为5.2%，尽管通NEON指令集可以将模乘运算的速度提升86.2%，但此技术在点加、倍点及SM2签名运算上的速度提升却较为有限。造成这一差异原因首先是并行化局限性，点加和倍点运算包含多个序列依赖步骤，这限制了并行执行的可能性。NEON指令集优化虽然可以加速独立的数据操作，但在算法中存在的数据依赖导致并行化的效果受限。其次也是最主要的原因便是点加和倍点运算需要在执行前后转换数据格式以适应NEON指令集，这一转换过程产生了额外的计算开销。

表5 点运算优化与原始GmSSL库实现的时钟周期对比

运算操作	时钟周期数		提升比率
	GmSSL[1]	点运算优化	
点加	6938	6502	6.7%
倍点	5488	5030	9.1%
标量乘	2 297 357	2 127 384	8.0%
SM2签名	2 739 607	2 604 049	5.2%

我们展示了并行计算的过程中GmSSL库与NEON指令集之间的数据类型转换结果，如图4所示。在GmSSL中每个256位数据的数据均存储在8×64位的数组中，每个数组元素只有低32位存储有效数据，在图4中用深色方块表示有效数据存储的部分。而由于NEON指令集的向量寄存器最大宽度为128位，考虑乘法运算存在比特位溢出，每次并行操作仅限于处理一对32位数据。这一限制意味着，我们必须将8×64位的数据存储格式转换为8×32位更细粒度的数据格式，以适应NEON的并行处理要求。此数据转换不仅涉及位宽的调整，也包括在NEON优化计算前后对数据格式的重新排列。原

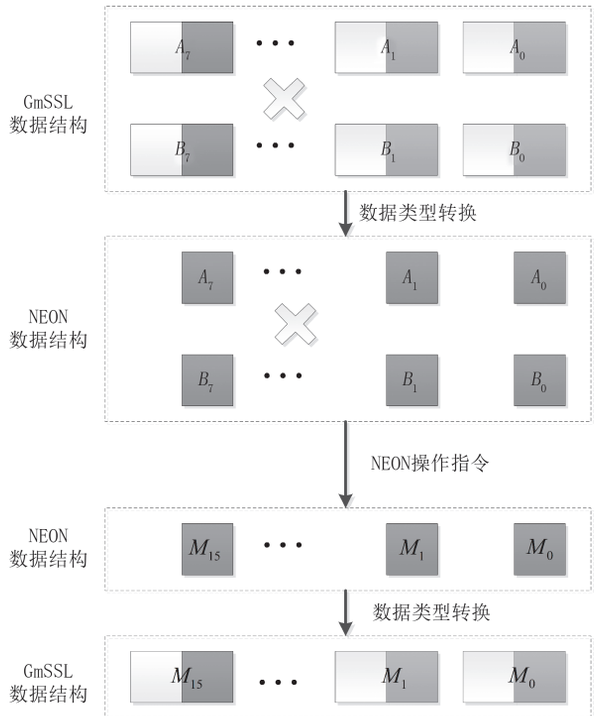

图4　GmSSL与NEON之间的数据转换

始数据格式必须从64位转换为32位才能加载到NEON的向量寄存器中，完成并行计算后又需将结果转换回64位格式。这种双向转换过程导致了显著的性能损耗，尤其是在需要频繁进行此类转换的点加和倍点运算中。因此，尽管NEON指令集在模乘和模平方运算中表现出了良好的性能提升，但在整个点加和倍点运算中的性能增益却受到了严重制约。

值得注意的是，单独对模乘和模平方运算进行性能测量时，NEON指令集展现出了较为理想的性能提升，这进一步证明了NEON并行计算在密码学应用中的潜在可行性。我们的研究表明，虽然当前NEON指令集的应用在GmSSL库中受到数据结构转换的限制，但通过针对密码学特有的计算特征和数据处理需求来优化NEON的应用策略，有望在未来实现更加高效的密码库。

在标量乘运算层面，我们测量了NAF算法与原始GmSSL中的传统二进制展开算法所消耗的时钟周期数，如表6所示。基于NAF展开的标量乘算法相比原始GmSSL库中二进制展开实现的时钟周期数由2 297 357降至1 943 209，速度提升了18.2%，对于SM2签名算法速度提升了12.8%，这一提升在2.4节中的理论分析中已有预期。NAF算法通过减少倍点运算的数量，优化了标量乘算法的时间复杂度。

表6　标量乘运算优化与原始GmSSL库实现的时钟周期对比

运算操作	时钟周期数		提升比率
	GmSSL[1]	标量乘运算优化	
标量乘	2 297 357	1 943 209	18.2%
SM2签名	2 739 607	2 428 375	12.8%

不同优化方案在实现标量乘和SM2签名算法时所消耗的时钟周期数如图5所示。图5中横轴分别对应本文所提出的三种优化方法以及这三种方法的综合优化结果，而虚线表示不同优化方法相较于原始GmSSL中签名算法的提升比。首先，域运算优化实现了最高达76.3%的速度提升。由于模乘运算是SM2算法中底层域运算操作的核心组成部分，这种优化直接显著降低了运算的时钟周期数。这表明，模乘和模平方等域运算在密码算法中的重要性，同时也表明优化策略应当集中于算法的底层运算，尤其是那些在算法执行中频繁出现的部分。其次，在点加和倍点运算方面，图5显示了相对较低的速度提升比率，对于SM2签名算法，速度提升比率为8.0%。相比于域运算的优化，这一提升幅度较低，反映了尽管NEON指令集可以提供一定程度的优化，但其效果受限于算法中的数据依赖性和并行化局限性。再次，通过引入NAF算法来优化标量乘运算，减少了所需的时钟周期数，具体来说，这种优化方法使得运算速度提升了18.2%，这一提升率介于点运算优化和域运算优化之间。综合本文提出的三种优化方法，签名算法的速度相较于原始GmSSL提升了118.3%。

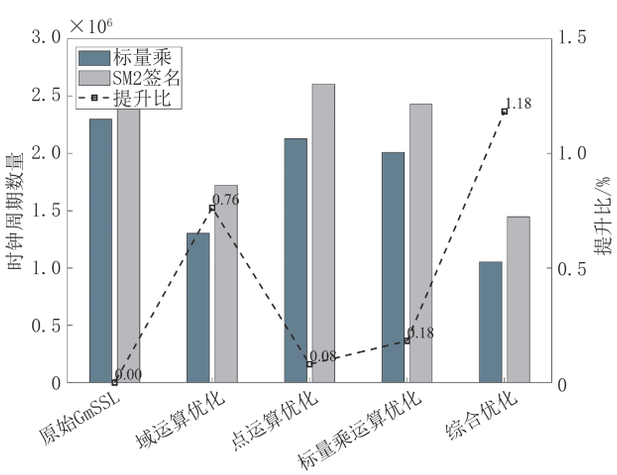

图5　不同优化算法提升比率

最后我们提供了一个全面的性能比较分析如表7所示，其中本文指代综合三种优化方法实现的SM2算法。我们分别展示了本文实现的SM2算法

表7 SM2算法实现时钟周期对比

运算操作	时钟周期数			提升比率
	GmSSL[1]	OpenSSL[2]	本文	
模乘/平方	580	293	261	122.2%/12.3%
点加	6938	5717	3493	98.6%/63.7%
倍点	5488	4748	2840	93.2%/67.2%
标量乘	2 297 357	2 119 235	1 052 182	118.3%/101.4%
SM2签名	2 739 607	2 254 923	1 447 182	89.3%/55.8%

与原始GmSSL库以及OpenSSL库在实现速度上的差异。其中，OpenSSL库在域运算层采用Montgomery约减算法，算法将素数域上的点转换到Montgomery域，从而有效避免了模约减运算。而在点运算层，需要将Montgomery域上的点重新转换回素数域，以便执行点加和倍点运算。因此，OpenSSL库与本文存在算法实现上的差异，为了保证实验的客观和公平性，本文没有将不同运算层面的优化实现结果与OpenSSL库进行比较，而只展示了综合优化后的不同运算操作与OpenSSL库在实现速度上的差异。具体来讲，对于模乘运算，本文相对于GmSSL的提升比率为122.2%，但相对于OpenSSL的提升比率仅有12.3%，这说明原始的OpenSSL库对于模乘运算的实现方案就是较优的。对于点加和倍点运算，本文相对于GmSSL的提升比率为98.6%和93.2%，相对于OpenSSL也提升了63.7%和67.2%。这表明本文提出的优化方法对两种密码库点运算操作的速度都有较大的提升。对于标量乘算法，本文相对于GmSSL速度提高了118.3%，相对于OpenSSL速度提高了101.4%。对于SM2签名算法，本文相对于GmSSL速度提高了89.3%，相对于OpenSSL速度提高了55.8%。综合以上分析，可以看出本文提出的算法在不同运算层面中都显著减少了所需的时钟周期数，从而大大提高了标量乘算法的实现速度，这对于提升椭圆曲线密码体制的整体性能至关重要。

5 总结与展望

本文对GmSSL3中的SM2算法进行了深入优化，从域运算层到标量乘运算层，实现了全方位的性能提升。实验数据表明，通过引入快速模约减算法、使用NEON指令集并行处理，以及采用NAF展开法等多项优化方法，本文实现的SM2算法在性能上相比原始GmSSL有显著提升。与主流密码算法库OpenSSL相比，本文提出的实现方案在多个关键操作上均表现出更优的性能。其中，对于标量乘算法本文与GmSSL相比速度提高了118.3%，与OpenSSL相比速度提高了101.4%。这一结果不仅证明了优化方案的有效性，也凸显了本研究在密码算法优化领域的实际应用价值。

本文的优化方法有望进一步推广至更多的密码算法和密码库中，以适应不同的硬件架构和应用需求。考虑到多样化的应用场景和不断演进的硬件技术，未来的研究可以集中在以下几个方面：

(1)跨平台性能优化：探索本文优化策略在其他硬件平台上的应用潜力，如云服务器、移动设备等，以提供更广泛的兼容性和适应性。

(2)算法泛化研究：将本研究的优化方法应用于其他椭圆曲线密码算法，甚至是非椭圆曲线的密码算法，以验证优化策略的泛化性。

(3)硬件优化的深入研究：针对特定硬件特性的深度优化，例如探索更高级的并行处理技术和硬件加速器。

(4)安全性与性能的平衡：在追求算法性能提升的同时，保持对算法安全性的严格要求，确保优化措施不会引入安全隐患。

总而言之，本文的工作不仅提供了一种更高效、更快速的SM2算法实现，还为国内网络安全领域的进一步发展提供了有力支持。本文的研究成果将有助于增强国内数据安全，推动信息技术的可持续发展，并为满足特定的国家安全需求提供了强大的密码学工具。

参 考 文 献

[1] Guan Z. GmSSL. v3.1.0. https://github.com/guanzhi/GmSSL. 2023

[2] OpenSSLSoftwareFoundation. OpenSSL. v3.1.2. https://github.com/openssl/openssl. 2023

[3] Chinese Encryption Administration. GM/T 0003-2012 SM2 Elliptic Curve Public-Key Cryptography Algorithm. Beijing: China Standard Press, 2010

[4] Karatsuba A. Multiplication of multidigit numbers on automata. Soviet Physics Doklady, 1963, 7: 595-596.

[5] Montgomery P L. Modular multiplication without trial division. Mathematics of Computation, 1985, 44(170): 519-521.

[6] Solinas J A. Generalized mersenne numbers. Faculty of Mathematics, University of Waterloo, Canada 1999.

[7] Hankerson D, Menezes A. Elliptic curve cryptography. Encyclopedia of Cryptography, Security and Privacy. Berlin, Germany: Springer, 2021: 1-2.

[8] Pabbuleti K C, Mane D H, Desai A, et al. SIMD acceleration

of modular arithmetic on contemporary embedded platforms// IEEE High Performance Extreme Computing Conference. Waltham，USA，2013：1-6.

[9] Wang W，Wang W，Lin J，et al. SMCOS：Fast and parallel modular multiplication on ARM NEON architecture for ECC// Information Security and Cryptology：17th International Conference，Inscrypt 2021. Virtual Event，2021：531-550.

[10] Miller V S. Use of elliptic curves in cryptography//Proceedings of the Conference on the Theory and Application of Cryptographic Techniques.Berlin，Germany，1985：417-426.

[11] Márquez R C，Sarmiento A J C，Solano S S. Speeding up elliptic curve arithmetic on ARM processors using NEON instructions. Revista Científica de Ingeniería Electrónica，Automática Comunicaciones，2020，41(3)：1-20.

[12] Van Luc P，Hai H D，Tan L D. Improving the Efficiency of Point Arithmetic on Elliptic Curves Using ARM Processors and NEON. International Journal of Network Security，2022，24(2)：364-376.

[13] Montgomery P L. Speeding the Pollard and elliptic curve methods of factorization. Mathematics of computation，1987，48(177)：243-264.

[14] Kocher P，Jaffe J，Jun B. Differential power analysis// Proceedings of the 19th Annual International Cryptology Conference. Santa Barbara，USA，1999：388-397

[15] Cohen，Henri，et al.，eds. Handbook of elliptic and hyperelliptic curve cryptography. Boca Raton，USA：CRC Press，2005

[16] Mahdavi R，Saiadian A. Efficient scalar multiplications for elliptic curve cryptosystems using mixed coordinates strategy and direct computations//Proceedings of the 9th International Conference on Cryptology and Network Security. Kuala Lumpur，Malaysia，2010：184-198.

[17] Zhao Z，Bai G. Ultra high-speed SM2 ASIC implementation// Proceedings of the 2014 IEEE 13th International Conference on Trust，Security and Privacy in Computing and Communications. Beijing，China，2014：182-188.

[18] Huang J，Liu Z，Hu Z，et al. Parallel implementation of SM2 elliptic curve cryptography on Intel processors with AVX2// Proceedings of the 25th Australasian Conferenceon Information Security and Privacy. Perth，Australia，2020：204-224.

[19] Gordon D M. A survey of fast exponentiation methods. Journal of algorithms，1998，27(1)：129-146.

[20] Faz-Hernández A，López J，Dahab R. High-performance implementation of elliptic curve cryptography using vector instructions. ACM Transactions on Mathematical Software (TOMS)，2019，45(3)：1-35.

QIAO Han，Ph. D. candidate. His research interest is real-time systems scheduling.

WANG An，Ph. D.，professor. His research interests include side-channel analysis and countermeasures.

WANG Bo，Ph. D.，associate professor. Her research interests include hardware security and processor security，software-defined chip.

SU Chang-Shan，B. S.，research assistant. His research interests sinclude cryptographic algorithms and hardware/software secure implementation.

LI Gen，Ph. D.，professor. His research interests include computer architecture and microprocessor design.

TANG Yu-Xing，Ph. D.，professor. His research interests include computer architecture and microprocessor design.

ZHU Lie-Huang，Ph. D.，professor. His research interests include cryptography，network and information security.

Background

This paper delves into information security，specifically focusing on optimizing cryptographic algorithms. In the era marked by the advent of the internet of things and digitalization，the importance of information security has escalated significantly. Adequate data protection becomes paramount，especially in scenarios involving extensive interconnectivity and data exchange among various user devices. Cryptographic algorithms play a crucial role in safeguarding sensitive data and communications. With technological advancements，optimizing the implementation of these algorithms has emerged as a critical issue. Significant progress has been made in the research and optimization of cryptographic algorithms. However，challenges

and opportunities for improvement, especially regarding efficiency and security, still exist.

This research focuses on the SM2 algorithm within the GmSSL3, a domestically developed commercial cryptographic open-source library. Spearheaded by Professor Guan Zhi from Peking University, GmSSL3 has significantly improved functionality compared to its predecessor. However, compared to international cryptographic libraries, GmSSL3 exhibits certain deficiencies in implementation speed. This study aims to improve the efficiency of implementing the SM2 algorithm in GmSSL. The optimizations encompass three primary aspects. Firstly, a rapid modular reduction algorithm is introduced at the domain operation level to increase the speed of modular multiplication operations. Secondly, NEON instruction sets are used to parallelize point addition and doubling operations at the point operation level, effectively accelerating these processes. Thirdly, the NAF method is utilized at the scalar multiplication layer instead of traditional binary expansion. This approach decreases the number of point additions, thus enhancing the speed of scalar multiplication algorithms.

Through these comprehensive optimization measures, the performance of the SM2 algorithm in the GmSSL library has been significantly enhanced, achieving a 118.3% improvement in efficiency. This achievement demonstrates the effectiveness of the optimization strategies in improving the performance of the SM2 algorithm. It provides valuable insights and guidance for researchers aiming to optimize elliptic curve cryptographic algorithms in other libraries. The results of this study suggest that targeted optimization strategies can significantly improve the efficiency of cryptographic algorithms without compromising security and functionality, offering substantial theoretical and practical significance in information security. Professor Wang An's research is supported by Project for Reconstruction of Industrial Foundation and High Quality Development of Manufacturing Industry (No. 0747-2361SCCZA193)