毕业论文（设计）

论文（设计）题目：素域上 Kobitz 曲线上 GLV 归约方法研究

姓　　名　耿敏祺

学　　号　202100161160

学　　院　网络空间安全学院

专　　业　密码科学与技术

年　　级　2021 级

指导教师　许光午

2025 年 5 月 16 日

摘　　要

在椭圆曲线密码学中，标量乘法运算的效率直接影响着密码算法的整体性能。Koblitz 曲线因其独特的数学结构，在现代密码学中占据重要地位。素域上的 Koblitz 曲线通过合理的算法设计可以实现高效的标量乘法计算，其中 GLV 方法（Gallant-Lambert-Vanstone Method）作为一种经典优化技术，其核心思想是通过将标量分解，再利用可高效计算的自同态映射来提高计算效率。本论文围绕 GLV 方法的归约步骤展开研究，对于大特征域上的特殊 Koblitz 曲线（如 secp256k1），利用 Eisenstein 整数环的性质，能够推导出适用于标量乘法的有效分解公式。基于该方法，本文在 C 环境下实现了 secp256k1 曲线标量分解的核心模块，创新性地采用了一些预计算优化技巧，通过理论分析证明了分解得到的子标量不超过 128 比特，并通过实验验证了分解结果的有效性。

关键词：椭圆曲线密码学；标量乘法优化；GLV 方法；secp256k1 曲线

Abstract

In elliptic curve cryptography, the efficiency of scalar multiplication directly determines the overall performance of cryptographic algorithms. Koblitz curves, distinguished by their unique mathematical structure, hold significant importance in modern cryptography. For Koblitz curves defined over prime fields, well-designed algorithms can achieve highly efficient scalar multiplication. Among optimization techniques, the Gallant-Lambert-Vanstone (GLV) method stands out as a classical approach, leveraging scalar decomposition and efficiently computable endomorphisms to enhance computational performance. This research focuses on the reduction step of the GLV method. For special Koblitz curves over large-characteristic fields (e.g. secp256k1), we derive an efficient scalar decomposition formula by utilizing properties of the Eisenstein integer ring . Based on this method, this paper implements the core module for scalar decomposition of the secp256k1 in C, incorporating innovative precomputation optimization techniques. In this paper, it is proved that the size of the subscalar is less than 128 bits by theoretical analysis, and the validity of the decomposition result is verified by experiments.

Keywords: elliptic curve cryptography, scalar multiplication optimization, GLV method, secp256k1

目　　录

第 1 章 前言

1.1 研究背景

在现代密码学领域，椭圆曲线密码学（ECC）因其高效的密钥长度和强大的安全性，已成为公钥密码系统中不可或缺的一部分。椭圆曲线密码学的核心操作是椭圆曲线上的标量乘法，其计算效率直接影响整个密码算法的性能。Koblitz 曲线，作为椭圆曲线的一种特殊形式，因其独特的数学结构和高效的计算特性，在密码学应用中占据重要地位 [1, 2]。其中，二进制域上的 Koblitz 曲线（定义在$\mathbb{F}_{2^m}$域上，标准方程为 $y^2 + xy = x^3 + ax^2 + 1, a \in \{0,1\}$。）凭借其 Frobenius 自同态特性可实现极高效的 $\tau - adic$ 标量乘法 [3, 4]，在资源受限环境中展现出卓越性能，特别适合硬件加速，被广泛应用于 NIST 标准的 K-系列曲线中 [5]；而素域上的 Koblitz 曲线（定义于 $\mathbb{F}_p$ 域，如比特币采用的 secp256k1 曲线）则利用复数乘法自同态和 GLV 方法（Gallant-Lambert-Vanstone Method）[6]，在保持高安全性的同时显著提升计算效率，因其更好的通用性和兼容性在大规模商业应用中占据主导地位，成为区块链系统和传统 PKI 基础设施的首选方案 [7, 8]。

GLV 方法在大特征域上的特殊 Koblitz 曲线上表现突出，其核心思想是通过将标量进行分解，并利用可高效计算的自同态映射来提升计算效率，该方法实现简洁，性能优越，有广泛的应用。尽管 GLV 方法在理论上已经相对成熟，但在实际应用中，如何进一步优化其步骤以适应不同的计算环境和需求，仍然是一个值得深入研究的课题。特别是在区块链等需要高效计算的应用场景中，如何在保证安全性的前提下，进一步提升计算效率，是当前研究的热点之一。

1.2 本文的主要贡献

本论文围绕 GLV 方法的归约步骤展开研究，旨在通过深入分析和优化，提升大特征域上 Koblitz 曲线标量乘法的计算效率。具体而言，本文的主要贡献包括以下几个方面：

（1）理论分析与公式推导：本文详细分析了 GLV 方法在大特征域 Koblitz 曲线上的应用，分析了 GLV 方法标量分解的归约过程，特别是针对 secp256k1 曲线，

利用曲线中的特殊算术性质，能够推导出适用于标量乘法的有效分解公式 [9]，本文给出了公式分解结果上界的证明。这些公式不仅在理论上具有创新性，而且在实际计算中能够显著降低计算复杂度。

（2）算法实现与优化：基于上述理论分析，本文在 C 语言环境下实现了 secp256k1 曲线标量分解的核心模块。在实现过程中，创新性地采用了一些预计算优化技巧，使得在预先计算某些中间结果的过程中，在减少计算的开销的同时不丢失运算结果的精确性，从而进一步提高了算法的执行效率。

（3）实验验证：为了验证所提出方法的正确性和效率，本文进行了一系列实验。验证结果表明该方法实现的标量分解符合要求，本文测试了 GLV 优化方案在不同情况下标量乘法以及标量分解的效率，优化算法在计算效率上有显著提升。这不仅为 secp256k1 曲线在区块链等实际应用中的高效计算提供了可行的解决方案，也为其他类似曲线的优化提供了参考依据。

第 2 章 预备知识

2.1 椭圆曲线密码学概述

椭圆曲线密码学的发展源于两个重要的数学突破：1985 年，数学家 Neal Koblitz 和 Victor Miller 分别独立提出了将椭圆曲线理论应用于密码学的构想 [10, 11]。这一创新性地将代数几何中的椭圆曲线理论与现代密码学相结合，开创了公钥密码学的新纪元。ECC 的出现有效解决了传统公钥密码体制（如 RSA）在资源受限环境下的应用瓶颈。

从数学本质来看，ECC 是基于有限域上椭圆曲线有理点构成的阿贝尔群构造的密码系统。其核心安全假设建立在对椭圆曲线离散对数问题（ECDLP）计算复杂度的评估上。相较于有限域离散对数问题，ECDLP 在相同安全强度下能够实现更小的密钥尺寸。具体而言，160 位的 ECC 密钥提供的安全强度相当于 1024 位的 RSA 密钥，而 256 位的 ECC 密钥则相当于 3072 位的 RSA 密钥 [12, 13]。这种密钥效率的优势使得 ECC 在移动通信、物联网等资源受限场景中展现出巨大应用价值。ECC 的安全性和效率高度依赖于椭圆曲线的选择，需要综合考虑曲线的数学特性、实现效率和安全性证明。这也是 NIST 等标准化组织制定标准曲线列表的重要原因。典型的应用曲线包括 secp256k1、NIST P-256 等 [5]，这些曲线都经过严格的安全评估和性能测试。

数学基础与群结构：

椭圆曲线密码学的数学基础涉及多个层次的代数结构。首先，定义在有限域$\mathbb{F}_q$上的椭圆曲线 E 由广义 Weierstrass 方程描述 [4]：

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6。$$

其中系数 $a_i \in \mathbb{F}_q$。在实际密码应用中，通常会使用简化形式的方程：在素域 $\mathbb{F}_q$ 上简化为：$y^2 = x^3 + ax + b$，在二进制域 $\mathbb{F}_{2^m}$上简化为：$y^2 + xy = x^3 + ax^2 + b$。曲线上的有理点与无穷远点 O 构成一个有限交换群，这个群结构是 ECC 的数学基础。群运算的基本操作是点加法运算，即给定两个相异点 P、$Q \in E$，计算其和 $R = P + Q$。群运算的几何意义可以通过弦切线法则来理解：两个点的和是通过这两点的直线与曲线的第三个交点关于 x 轴的对称点。这种几何解释虽然直观，但在实

际计算中需要转换为代数运算，为此可以使用不同的坐标系统（如雅可比坐标）以提高计算效率。需要注意，椭圆曲线群的阶（即曲线上点的数量）是一个重要的安全参数。

而标量乘法是指给定整数 k 和点 $P \in E$，计算 $kP = P + P + \ldots + P$（k 次），是 ECC 中最核心的运算，也是密码算法的主要计算瓶颈。为了提高计算效率，研究者们开发了多种优化技术，包括采用高效的标量表示方法（如 NAF 表示 [14]）优化计算路径，以及利用曲线本身的特殊算数性质 [1, 2] 等。

安全理论基础：

椭圆曲线密码学的安全性建立在三个核心计算难题之上：其中最难的是离散对数问题（DLP，或椭圆曲线上 ECDLP），即给定曲线上的两点 P、$Q = kP$，在已知 P、Q 的情况下求解整数 k。这个问题被认为是计算上不可行的，特别是当基点P的阶是一个大素数时。目前已知最好的通用解法是 Pollard's rho 算法，其时间复杂度为 $O(\sqrt{n})$，其中 n 是基点 P 的阶。这意味着要保证 128 位的安全强度，需要选择阶约为 2^{256} 的曲线。对于计算性 Diffie-Hellman 问题（CDHP），是指给定 P、aP、bP，计算 abP。这个问题是许多密钥协商协议（如 ECDH）的安全基础。在某些特殊情况下 CDHP 可能比 ECDLP 更容易，但对于特定的曲线，这两个问题被认为具有相当的难度。最后一个是判定性 Diffie-Hellman 问题（DDHP）：给定 P、aP、bP、cP，判定是否 $c \equiv ab \bmod n$。

ECC 的安全性不仅依赖于这些抽象的计算难题，还与具体的曲线参数选择密切相关。历史上曾出现过“不安全曲线”的案例，例如某些具有特殊性质的曲线可能存在 MOV 约化等攻击途径 [15]。因此，在实际应用中必须使用经过充分分析的标准化曲线。此外，实现层面的安全性也同样重要，包括防护侧信道攻击、确保随机数的质量等。

2.2 雅可比坐标

雅可比坐标系是一种通过引入额外坐标 Z 来优化椭圆曲线上的点乘操作的坐标系统。通过在标准笛卡尔坐标系 (x, y) 的基础上增加一个额外的坐标 Z，雅可比坐标能够减少计算过程中的除法操作，从而提高计算效率。在雅可比坐标系中，

椭圆曲线的方程被表示为：

$$Y^2Z = X^3 + aXZ^2 + bZ^3。$$

这里的 X, Y, Z 是坐标，a, b 是椭圆曲线的参数。

雅可比坐标下的点加法：

对于两个点 $P = (X_1, Y_1, Z_1)$ 和 $Q = (X_2, Y_2, Z_2)$ 的加法，雅可比坐标的加法公式如下：

（1）计算中间量：

$$U_1 = X_1Z_2^2,\ U_2 = X_2Z_1^2,$$

$$S_1 = Y_1Z_2^3,\ S_2 = Y_2Z_1^3,$$

$$H = U_2 - U_1,\ R = S_2 - S_1。$$

（2）计算新的坐标：

$$X_3 = R^2 - H^3 - 2U_1H^2,$$

$$Y_3 = R(H^2 - X_3) - S_1H^3,$$

$$Z_3 = Z_1Z_2H。$$

雅可比坐标下的倍加：

椭圆曲线上的倍加操作（即计算 $2P$，P 是椭圆曲线上的点）也是通过雅可比坐标进行优化的。倍加操作的关键是通过重复应用点加法公式来计算。对于点 $P = (X, Y, Z)$，倍加操作可以通过以下公式进行：

（1）计算中间量：

$$A = 3X^2 + aZ^4,\ B = Y^2Z^2,\ C = 2B,$$

$$D = A^2 - 2C,\ E = 2(AX - D),\ F = E^2 - 8D,$$

$$G = 2YZ^3,\ H = A(B - D) - 2F。$$

（2）计算新的坐标：

$$X_3 = F,\ Y_3 = H,\ Z_3 = 2Z^3。$$

雅可比坐标的优势在于，以上公式只涉及加法和乘法，没有除法操作，这使得计算更加高效 [16]。此外，雅可比坐标还可以减少内存使用，因为它只需要存储一个额外的坐标 Z，而不需要存储多个中间结果。在雅可比坐标的计算点加和倍加的基础上，可以采用一些优化技巧来加速标量乘法的计算。

2.3 标量乘的一般加速方法

标量乘法（点乘）基于椭圆曲线上的一个点和一个标量来运算。标量乘法的核心操作是通过重复的点加法来实现。为了提高这些运算的效率，尤其是当标量较大时，通常需要使用一些技巧和优化方法。

预计算与窗口法：

预计算是一种常见的优化技术 [17]，尤其在固定点已知的情况下。对于标量乘法，预计算方法通常通过表格的形式将点的加和倍加运算结果存储在内存中，这样可以避免在实际计算时重复进行同样的运算。通过预先生成的点值表，能够在无需重复计算的前提下，快速构造出 kG 的结果。结合窗口法，可以进一步减少所需的点加和点倍加次数。

窗口法是将标量 k 分组为固定大小的“窗口”，每个窗口对应一组预计算点，从而避免传统二进制展开中每个位都要进行一次倍加的低效率方式。可以预先计算出所有可能窗口值对应的点值（固定窗口法），也可以动态选择非零位段，跳过零，提高密度（滑动窗口法）。通过合理选择窗口大小，可以平衡存储需求与计算效率。较大的窗口可以减少点加的次数，但需要更多的存储空间来存储预计算表格。在雅可比坐标系下，由于点加和点倍加运算不涉及除法，窗口法的每一步操作都可以非常高效地执行。因此这两种方法结合后，在实际中常常构成 ECC 实现的主流策略。

指数重编码技术（Exponent Recoding Techniques）：

指数重编码方法的核心思想是改变标量 k 的表达方式，使其在展开时尽可能包含更少的非零位，从而减少点加操作 [14]。最常见的方法是使用非相邻表示（Non-Adjacent Form, NAF）。

NAF 保证了连续两个非零位之间总有一个零，从而降低了点加操作频率，其平均非零位密度约为传统二进制表示的一半。此外，进一步的窗口 NAF 方法允许每个非零位段跨越多个二进制位，并配合预计算进一步减少操作数。在特定类型的椭圆曲线中，还可以采用 τ-非相邻表示（TNAF, τ-adic Non-Adjacent Form）进行优化。TNAF 是 NAF 在 τ-adic（τ进制）扩展下的推广形式，例如利用二进制 Koblitz 曲线上的 Frobenius 映射（记作 τ）的高效实现 [3]，将标量表示为 $k = \sum u_i \tau^i$ 的稀疏形式，其中 $u_i \in \{0, \pm 1\}$，并满足非零系数之间间隔至少一个零位。由于 τ 的幂

次可通过 Frobenius 映射高效计算，这样能进一步减少点加操作的实际代价，TNAF 表示保持了 NAF 的稀疏特性，与窗口法结合，在 Koblitz 曲线实现中表现出色。

加法链：

加法链方法通过构造特定的整数加法顺序，减少重复的乘法操作 [18]。例如，对于固定的 k，可构造一条“最短路径”将 k 分解为一系列加法与倍加步骤。其典型场景是 RSA 中固定私钥指数的优化，在椭圆曲线中，若某些常见的标量固定或可预测，同样适用。

同时多重指数运算（Simultaneous Multiple Exponentiation）：

这种方法的目标是在多个点和标量同时参与时，合并其计算路径以提高效率。其典型形式是：$k_1Q_1 + k_2Q_2 + \cdots + k_tQ_t$。在 ECC 中，尤其在签名验证（如 ECDSA 验证）时，该形式广泛存在。在验证过程中需要同时计算两个标量乘积的和。如果采用逐个点乘再相加的方式，效率非常低。而多重指数运算技术通过“Shamir’s Trick”的合并，同时处理两个或多个标量与点乘，有效压缩点倍加操作。Shamir’s Trick 的核心思想是通过将两个标量的位同时处理，合并倍加路径，从而节省时间。比如我们要计算 $k_1P + k_2Q$，使用 Shamir’s Trick，我们把 k_1 和 k_2 的二进制表示打包处理，例如：

表 2-1 Shamir’s Trick 的位运算表

位数	$k_1[i]$	$k_2[i]$	操作
3	1	1	加$P + Q$
2	1	0	加P
1	0	1	加Q

预计算 4 个组合：O，P，Q，$P + Q$。然后每一轮只需根据 $(k_1[i], k_2[i])$，来查找表 2-1 并加上对应点，同时每轮做一次点倍加（相当于向高位移动）。总共只需要一次循环、一次倍加链、少量点加操作。整体效率比两次单独计算再相加快很多。

而 GLV 方法通过引入一个高效的自同构映射 ϕ，将标量 k 分解为 $k_1 + k_2\lambda$，然后转化为两个标量乘法之和 $kP = k_1P + k_2\phi(P)$，正是上述方法的一个特殊应用，结合高效的 ϕ 映射达到更佳的运算效果。

2.4 椭圆曲线的自同态映射

在椭圆曲线密码学中，自同态（endomorphism）是一个非常重要的数学工具。简单地说，自同态是定义在椭圆曲线上的一种映射，它既保持曲线的几何结构，也保持加法运算的代数结构 [19]。GLV 方法之所以能够显著加速标量乘法，正是基于对这些结构性映射的利用。

2.4.1 定义

设 E 是定义在有限域 $\mathbb{F}_q$ 上的椭圆曲线，自同态是指如下的映射：

$$\phi: E \to E。$$

满足两个条件：

（1）代数封闭性：对任意 $P, Q \in E$，有 $\phi(P+Q) = \phi(P) + \phi(Q)$。

（2）恒等元素保持：$\phi(O) = O$，其中O是椭圆曲线上的无穷远点（加法单位元）。

换言之，ϕ 是保持椭圆曲线加法运算性质的映射，是一个群自同态（group homomorphism）。

2.4.2 可高效计算的自同态

所有定义在 E 上的自同态组成一个环（endomorphism ring），记为 $End(E)$，称为椭圆曲线的自同态环。对于大多数椭圆曲线，$End(E) = \mathbb{Z}$，也就是说，只有“倍数映射”如 $[m]P$ 才是自同态。对于具有复杂乘法结构（CM，Complex Multiplication）的椭圆曲线，自同态环可能是一个更大的环，如：

$$End(E) \cong \mathbb{Z}[\sqrt{-d}],$$

或某个整数环 $\mathcal{O}_K$（某个二次虚数域 K 的整数环）。在这些曲线上，存在非平凡的自同态（非整数倍映射）。这类曲线是 GLV 方法的核心适用对象。

我们可以根据定义和可计算性，把椭圆曲线上常用的自同态大致分为以下几类：

（1）倍数映射（Multiplication-by-m map）：最基本的自同态是整数倍映射，即：

$$[m](P) = mP。$$

该映射在理论上属于自同态环$\mathbb{Z}$的一部分，在实际运算中构成标量乘法的基础。

（2）Frobenius 自同态（Frobenius Endomorphism）：适用于定义在特征 p 的有限域 $\mathbb{F}_q$ 上的曲线，其形式为：

$$\phi(x,y)=(x^q,y^q)。$$

这是一种特殊且高效的自同态，尤其在小特征域的 Koblitz 曲线上被广泛应用（如 Solinas 方法），因为其运算仅涉及域元素的幂次操作，等同于“移位”或“位运算”，在硬件上极易实现。

（3）可高效计算的特殊自同态：这类自同态是 GLV 方法所关注的重点。设 E 是定义在 $\mathbb{F}_q$ 上的椭圆曲线，$P\in E(\mathbb{F}_q)$ 是阶为素数 n 的点。若一条曲线 E 的自同态环包含非整数的元素（如复整数或更高代数数域中的元素），存在形如：

$$\phi(P)=\lambda P \ mod\ n, \lambda\in\mathbb{Z}_n \tag{1}$$

的自同态作用 ϕ。这种情形下，映射 ϕ 作用于 $\langle P\rangle$ 是乘法映射 $[\lambda]$，可用于将标量 k 分解为两个更短的子标量，从而降低乘法复杂度。

这里给出两个例子，参考自 [20]：

例 1 设 $p\equiv 1(\mathrm{mod}4)$ 是素数，考虑定义在 $\mathbb{F}_p$ 上的椭圆曲线

$$E_1: y^2=x^3+ax。$$

设 $\alpha\in\mathbb{F}_p$是 4 阶元。则映射 $\phi: E_1\to E_1$ 由 $(x,y)\mapsto(-x,\alpha y)$ 和 $O\mapsto O$ 定义，是定义在 $\mathbb{F}_p$ 上的自同态。如果 $P\in E(\mathbb{F}_p)$ 是阶为素数 n 的点，则 ϕ 作用于 $\langle P\rangle$ 是乘法映射 $[\lambda]$，即对于所有 $Q\in\langle P\rangle$，$\phi(Q)=\lambda Q$。其中 λ 是满足 $\lambda^2\equiv -1(\mathrm{mod}n)$ 的整数。这里 $\phi(Q)$ 可以仅使用 $\mathbb{F}_p$ 中的一次乘法计算。

例 2 设$p\equiv 1(\mathrm{mod}3)$ 是素数，考虑定义在$\mathbb{F}_p$上的椭圆曲线

$$E_2: y^2=x^3+b。 \tag{2}$$

设 $\beta\in\mathbb{F}_p$是 3 阶元。则映射 $\phi: E_2\to E_2$ 由 $(x,y)\mapsto(\beta x,y)$ 和 $O\mapsto O$ 定义，是定义在 $\mathbb{F}_p$ 上的自同态。如果 $P\in E(\mathbb{F}_p)$ 是阶为素数 n 的点，则 ϕ 作用于 $\langle P\rangle$ 是乘法映射 $[\lambda]$，即对于所有 $Q\in\langle P\rangle$，$\phi(Q)=\lambda Q$。其中 λ 是满足 $\lambda^2+\lambda\equiv -1(\mathrm{mod}n)$的整数。这里 $\phi(Q)$ 可以仅使用 $\mathbb{F}_p$ 中的一次乘法计算。

2.5 GLV 方法

GLV 方法的核心在于将一次传统意义上的高位标量乘法 kP 转换为两个较小标量的线性组合，从而显著降低整体的计算成本 [6]。这一思想建立在椭圆曲线上可高效计算的自同态映射 ϕ 的前提下。设 $E(\mathbb{F}_q)$ 是定义在有限域 $\mathbb{F}_q$ 上的椭圆曲线，$P \in E(\mathbb{F}_p)$ 是一个阶为素数 n 的点。存在一个 (1) 非平凡自同态映射 $\phi: E \to E$。其中 $\lambda \in \mathbb{Z}_n$ 是 ϕ 在 $\langle P \rangle$ 上的“特征值”，那么对任意标量 $k \in \mathbb{Z}_n$，我们都可以找到整数 $k_1, k_2 \in \mathbb{Z}_n$，使得：

$$k \equiv k_1 + k_2\lambda \ mod\, n。 \tag{3}$$

从而有：

$$kP = (k_1 + k_2\lambda)P = k_1P + k_2\lambda P = k_1P + k_2\phi(P)。$$

这个等式将原始标量乘法 kP 分解为了两个子问题：分别计算 k_1P 和 $k_2\phi(P)$，然后将它们相加即可得到最终结果。这种方法在数学结构上是完全等价的，但其计算复杂度却大大降低。关键原因在于：在适当的分解算法（后续将讨论）下，可以确保 k_1 和 k_2 位数（bit-length）显著小于 k，通常约为 $\frac{1}{2}log_2 n$。这意味着我们将一个高位标量乘法转化为了两个低位标量乘法，且可以采用多标量并行计算技术（如 Shamir's Trick）进一步提升效率。

此外，由于 ϕ 是一个高效可计算的自同态映射，计算 $\phi(P)$ 的成本远低于一次点乘，通常仅需常数次有限域乘法，或者一次符号或坐标变换。因此，将一部分运算从标量乘法“转移”到 ϕ 映射上，是一次成本显著优化的过程。这种降维策略尤其适合在大标量（如 256 位、384 位）情况下应用，其节省的运算量将更加显著；同时由于两个子标量乘法可以并行执行，也非常适合现代多核或硬件加速环境。

GLV 方法通过引入自同态映射，将一个计算成本高昂的点乘运算结构性地转化为两个低位运算的组合，从而实现了：（1）计算复杂度的下降；（2）并行计算的可能；（3）与已有加速技术（如同时多点乘法）综合。因此，这一标量分解思想构成了整个 GLV 方法的数学基础与效率来源。在接下来的内容中，本文将探讨如何通过算法手段高效求解 k_1, k_2，以及实际中如何实现该过程。

第 3 章 GLV 标量分解的归约方法

3.1 曲线选择与标量分解要求分析

关于曲线选择：

设 E 是定义在有限域 $\mathbb{F}_q$ 上的椭圆曲线，$P \in E(\mathbb{F}_q)$ 是一个阶为素数 n 的点。ϕ 是定义在 $\mathbb{F}_q$ 上的自同态。为了使 GLV 方法适用于该曲线，我们需要满足一个核心代数条件：自同态 ϕ 的特征多项式在模 n 意义下存在根 [6]。具体而言，设 ϕ 的特征多项式为 $f(x) = x^2 - Tr(\phi)x + Norm(\phi)$，其中 $Tr(\phi)$ 是 ϕ 的迹，$Norm(\phi)$ 是 ϕ 的范数。我们要求在模 n 下，存在一个整数 $\lambda \in \mathbb{Z}_n$，使得：

$$f(\lambda) \equiv 0 \ (mod\ n)。$$

换句话说，λ 是 ϕ 的一个模 n 根。在这种情形下，ϕ 在点 P 所生成的循环子群 $\langle P \rangle \subseteq E(\mathbb{F}_q)$ 上的作用等价于标量乘法 (1)，这正是 GLV 方法中进行标量分解的理论基础。由于 $f(x)$ 是一个二次多项式，在模素数 n 意义下是否存在根取决于其判别式 $D = Tr(\phi)^2 - 4Norm(\phi)$ 是否是模 n 的平方。根据模素数下二次剩余的概率分布，可以推断：大约一半的曲线满足该条件，即其自同态的特征多项式在模 n 下存在根。因此，从概率意义上说，GLV 方法并不依赖于极端稀有的曲线结构，而是具有较高的适用范围。

另一方面，从算法实用性的角度出发，是否采用 GLV 方法还取决于计算成本是否能带来实际的性能优势。在 GLV 原文 [6] 中，作者指出，如果自同态 ϕ 的计算成本低于约 $log_2 \frac{n}{3}$ 次点加倍操作的代价，那么该方法具有较强的实用价值。更具体地说，在实际应用中，如果 ϕ 的计算不超过 5 次点加倍的代价，就可以有效抵消标量分解与额外点加操作所引入的开销，带来显著的整体性能提升。这种衡量标准为在实际工程中选择是否使用 GLV 方法提供了可量化的参考依据。

特征值较大的素域 $\mathbb{F}_p$ 上的曲线现在具有实际意义。本文将讨论素域 $\mathbb{F}_p$ 上的曲线族 (2)，其形式为：

$$E_b: y^2 = x^3 + b/\mathbb{F}_p。$$

其中素数 $p \equiv 1 \ (mod 3)$，$b \in \mathbb{F}_p^*$。这一曲线族被称为 Koblitz 曲线，因为它是具有

简单表达式的 CM 曲线的特殊情况。对于素域 $\mathbb{F}_p$ 上的 Koblitz 曲线 E_b，当 $p \equiv 1(mod3)$ 时，可以使用 GLV 方法加速其标量乘法。通过适当选择 b 和 p，可以使椭圆曲线上有理点的个数 $\#E_b(\mathbb{F}_p)$ 为素数 [9]。在这种情况下，对椭圆曲线上任意点 $P \in E(\mathbb{F}_p)$，P 的阶为素数，我们仍将其记为 n，并且有 $n \equiv 1(mod3)$。存在 $\mathbb{F}_p$ 中的立方根单位 β 和 $\mathbb{F}_n$ 中立方根的 λ，使得对于每个点 $P \in E(\mathbb{F}_p)$，有：

$$\lambda P = (\beta x, y)。 \tag{4}$$

这意味着 λP（λ 被视为标量）可以高效地计算，成本为 $\mathbb{F}_p$ 中的一次乘法。

SECG（Standards for Efficient Cryptography Group）中描述的一种 Koblitz 曲线是：

$$\text{secp256k1: } y^2 = x^3 + 7/\mathbb{F}_p。 \tag{5}$$

其中 $p = 2^{256} - 2^{32} - 977$ 是一个 256 位的素数。这条曲线已被一些应用（如区块链平台如比特币的数字签名）选中，并且是 NIST 密码学基于离散对数的推荐曲线之一。本文的具体实现部分基于 secp256k1，在 C 语言环境下完成。

关于标量分解要求：

GLV 方法的关键步骤是找到满足 (3) 的两个较小整数 k_1, k_2，为了确保该分解在数学上正确、在计算中高效，需要满足以下几个核心要求：

（1）模 n 同余的正确性要求：首先必须保证标量分解在模 n 意义下是成立的，即 (3) 成立。这是 GLV 方法的基础，确保通过分解得到的两个子标量与原标量 k 之间的代数等价性，从而使得最终计算的点 kP 与 $k_1P + k_2\phi(P)$ 完全一致。

（2）子标量的“短度”要求：GLV 方法的最大优势在于通过结构性的分解，将一次高位的标量乘法转换为两个较短标量的运算。为了使分解具有实际的加速价值，k_1, k_2 的绝对值必须显著小于 k 本身。理想情况下，它们的位数应约为原始标量的一半。我们希望将一个 $log_2 n$ 位的整数拆解为两个约 $\frac{1}{2}log_2 n$ 位的整数，从而在点加和点倍加操作中减少所需次数。

（3）分解过程的高效性要求：在满足正确性与“短度”的前提下，GLV 方法还要求标量分解的过程本身应当计算量小、易于实现、可在加密系统中高效执行。否则，即便子标量变短，若分解过程耗时过高，也会抵消整体加速收益。

GLV 方法的理论基础表明，只要存在合适的 λ，并采用合理的分解策略，总可以保证分解后的 (k_1, k_2) 在大小上远小于原始标量，从而在总体复杂度上实现超过 50% 的加速潜力。

3.2 格上最近向量问题的归约

我们的问题可以简要表述如下：对输入：整数 n，整数 $\lambda, k \in \mathbb{Z}_n$，找到输出：$k_1, k_2$，满足 $k \equiv k_1 + k_2\lambda \ mod\, n$ (3)，且 (k_1, k_2) 尽可能小。

GLV 方法提出了一种利用格（lattice）理论进行标量分解的思路，其核心是将问题归约到格中的一个经典问题：最近向量问题（Closest Vector Problem, CVP）。基本思路如下：

首先我们知道向量 $(k, 0)$ 是满足条件 (3) 的一组 (k_1, k_2)，但我们期望向量 (k_1, k_2) 的欧几里得范数尽可能小。为此我们首先找到两个线性无关的短向量 $\mathbf{v_1}, \mathbf{v_2}$ 使其满足 (3)。然后在 $\mathbf{v_1}, \mathbf{v_2}$ 生成的整数格中找到一个接近 $(k, 0)$ 的向量 $\mathbf{v}$，也就是格上以 $(k, 0)$ 为目标点的 CVP 问题。令 $\mathbf{u} = (k, 0) - \mathbf{v}$， $\mathbf{u}$ 是一个短向量，且 $\mathbf{u}$ 同样满足条件 (3)。该方法通过构造模n映射的核格，并利用欧几里得算法和 Babai 近似算法 [21]，生成既满足模同余关系、又长度较短的分解结果。以下将结合原文中的具体推导，详细介绍分解过程。

3.2.1 构造核格

定义如下模 n 映射：

$$f: \mathbb{Z}^2 \to \mathbb{Z}_n, \quad f(i, j) = i + j\lambda \bmod n,$$

此映射的核为：

$$L := \ker(f) = \{(i, j) \in \mathbb{Z}^2 | i + j\lambda \equiv 0 \bmod n\}。$$

这是一个二维整数格。我们称其为“核格”，目标是在该格中找到两个线性无关的短向量 $\mathbf{v}_1, \mathbf{v}_2$，用于后续的格投影与标量分解。

3.2.2 使用扩展欧几里得算法构造格基

构造满足 $f(\mathbf{v}) = 0$ 的短向量 $\mathbf{v} = (r, -t)$ 的关键在于应用扩展欧几里得算法来求解 $\gcd(n, \lambda)$。由于 n 是素数，显然 $\gcd(n, \lambda) = 1$。算法产生一列等式：

$$s_i n + t_i \lambda = r_i,$$

其中初始条件为：

$$s_0 = 1, t_0 = 0, r_0 = n;\ \ s_1 = 0, t_1 = 1, r_1 = \lambda。$$

每一步迭代更新规则为：

$$r_{i+1} = r_{i-1} - q_i r_i。$$

并相应更新 s_i, t_i。该算法具有如下重要性质：

$$对所有\ i \geq 0，有：r_i > r_{i+1} \geq 0， \tag{6}$$

$$对 i \geq 1，有：|s_i| < |s_{i+1}|， \tag{7}$$

$$对 i \geq 0，有：|t_i| < |t_{i+1}|， \tag{8}$$

$$对 i \geq 1，有恒等式：r_{i-1}|t_i| + r_i|t_{i-1}| = n。 \tag{9}$$

当某个 $r_m \geq \sqrt{n}$ 且 $r_{m+1} < \sqrt{n}$ 时，令：

$$\mathbf{v}_1 = (r_{m+1}, -t_{m+1}),$$

由构造可知：

$$r_{m+1} + (-t_{m+1})\lambda = s_{m+1}n \equiv 0 \bmod n。$$

即 $f(\mathbf{v}_1) = 0$，因此 $\mathbf{v}_1 \in L$。同时，由于：

$$|r_{m+1}| < \sqrt{n},\ |t_{m+1}| < \sqrt{n} \Rightarrow ||\mathbf{v}_1|| \leq \sqrt{2n},$$

该向量长度满足“短向量”要求。

接下来，在 $\mathbf{v}_2 = (r_m, -t_m)$ 和 $\mathbf{v}_2 = (r_{m+2}, -t_{m+2})$ 中选择较短的向量作为 $\mathbf{v}_2$。这两个向量也属于核格，并与 $\mathbf{v}_1$ 线性无关。验证线性无关性的方法如下：如果 $\mathbf{v}_1$ 与 $\mathbf{v}_2$ 线性相关，例如：

$$\frac{r_{m+1}}{r_m} = \frac{t_{m+1}}{t_m},$$

但根据 (6), (8)，有：

$$\frac{r_{m+1}}{r_m} < 1,\ \frac{|t_{m+1}|}{|t_m|} > 1,$$

故两向量不可能线性相关。

此外，可以看到，$\mathbf{v}_1$ 与 $\mathbf{v}_2$ 仅依赖于 n 与 λ，与待分解的 k 无关。因此在实际实现中，这两个基向量可以预先计算并存储，无需在运行时重复计算。

3.2.3 最近格点投影与标量分解

构造完核格基底后，考虑如何将目标向量 $(k, 0)$ 在格中分解为：

$$(k_1, k_2) = (k, 0) - \mathbf{v},\ \mathbf{v} \in \mathrm{span}_{\mathbb{Z}}\{\mathbf{v}_1, \mathbf{v}_2\}。$$

将 $(k,0)$ 视为 $\mathbb{Q}^2$ 中的向量，可写作：

$$(k,0) = \beta_1\mathbf{v}_1 + \beta_2\mathbf{v}_2,\ \beta_1,\beta_2 \in \mathbb{Q},$$

然后将实数系数四舍五入为最接近的整数：

$$b_1 = \lfloor\beta_1\rceil,\ b_2 = \lfloor\beta_2\rceil,$$

构造格中向量：

$$v = b_1\mathbf{v}_1 + b_2\mathbf{v}_2,$$

最终得到标量分解：

$$(k_1,k_2) = (k,0) - \mathbf{v}。$$

这样简化的 Babai 方法 [21] 本质上是一种最近格点近似，只使用基本线性代数，无需复杂格归约，便于高效实现。

3.2.4 分解结果的范数界限

根据三角不等式与取整误差分析，我们可以做如下推导：设

$$\mathbf{u} = (k,0) - v = (k_1,k_2),$$

则 $\mathbf{u} = (\beta_1 - b_1)\mathbf{v}_1 + (\beta_2 - b_2)\mathbf{v}_2$，由于 $|\beta_i - b_i| \le \frac{1}{2}$，故：

$$||\mathbf{u}|| \le \frac{1}{2}||\mathbf{v}_1|| + \frac{1}{2}||\mathbf{v}_2|| \le \max(||\mathbf{v}_1||,||\mathbf{v}_2||)。$$

即最终分解 (k_1,k_2) 是长度不超过最大基向量长度的短向量，从而保证了子标量长度的稳定性。

3.3 Eisenstein 整数商选择的归约

3.3.1 有理点个数的表示

之前我们令椭圆曲线上有理点的个数 $\#E_b(\mathbb{F}_p)$ 为素数，并且有 $n \equiv 1(mod3)$。在 [22, 23] 中我们知道，每一个有理素数都可以表示为 Eisenstein 整数环 $Z[\omega]$ 某个元素 $\pi = c + d\omega$ 的范数 $N(c + d\omega)$。即对于给定的素数 $n \equiv 1(mod3)$，有

$$n = c^2 - cd + d^2, \tag{10}$$

其中 $c,d \in \mathbb{Z}$。对整数 λ，有：$1 + \lambda + \lambda^2 = 0(modn)$。因此有：

$$\begin{aligned}(c + d\lambda)(c + \lambda^2 d) &= c^2 + (\lambda + \lambda^2)cd + \lambda^3 d^2 \\ &\equiv c^2 - cd + d^2(modn) \equiv 0(modn),\end{aligned}$$

可以得到：

$$c + d\lambda \equiv 0(mod n)。$$

否则，如果 $c + \lambda^2 d \equiv 0(mod\ n)$，则 $c - d - d\lambda \equiv 0(mod\, n)$,可以用 $(c - d, -d)$ 替代 (c, d)。

因此，我们可以构造格：

$$L: = \{(x, y) \in \mathbb{Z}^2 | x + y\lambda \equiv 0 \bmod n\},$$

然后使用 Lagrange-Gauss 二维格算法 [24] 可以在多项式时间找到一个非零最短向量 $\mathbf{v} = (c, d) \in L$，得到 $n = c^2 - cd + d^2$。

3.3.2 子标量的构造

现在整数 n, c, d，整数 $\lambda, k \in \mathbb{Z}_n$已知，我们的目标是找到满足 (3) 的 (k_1, k_2)，且 (k_1, k_2) 尽可能小。首先，定义 α_1, α_2,令 q_1, q_2 为整数，设

$$\alpha_1 = \frac{k(c-d)}{n} - q_1,\ \alpha_2 = \frac{-kd}{n} - q_2,$$

也就是说将这两个分数的整数部分拆出来了一些，剩下的是 α_1, α_2。然后我们将 $k_1 + k_2\omega$ 写成如下形式：

$$k_1 + k_2\omega = (\alpha_1 + \alpha_2\omega)(c + d\omega)。 \tag{11}$$

通过构造这样的表示形式，我们可以将 k 做变换：

$$k = \frac{k}{c+d\omega}(c + d\omega) = \frac{k(c+\bar{\omega})}{n}(c + d\omega),$$

接着将之前 α_1, α_2 的表达带入，可以整理出：

$$\begin{aligned} k &= ((q_1 + q_2\omega) + (\alpha_1 + \alpha_2\omega))(c + d\omega) \\ &= (q_1 + q_2\omega)(c + d\omega) + (k_1 + k_2\omega) \\ &= q_1 c + k_1 + \omega(q_2 c + q_1 d + k_2) + \omega^2 q_2 d。 \end{aligned}$$

ω 是三次单位根，有：$\omega^2 = -\omega - 1$，因此

$$k = (q_1 c + k_1 - q_2 d) + \omega(q_2 c + q_1 d - q_2 d + k_2)。$$

我们得到：

$$q_1 c + k_1 - q_2 d = k,$$

$$q_2 c + q_1 d - q_2 d + k_2 = 0。$$

即：

$$k_1 = k + q_2 d - q_1 c,$$
$$k_2 = q_2 d - q_2 c - q_1 d。 \tag{12}$$

3.3.3 Eisenstein 整数商的选择与标量分解

由于 $k_1 + k_2\omega = (\alpha_1 + \alpha_2\omega)(c + d\omega)$，我们的问题归约为：通过选择合适的 Eisenstein 整数 $(q_1 + q_2\omega)$，使得 $k_1 + k_2\omega$的“大小”最小化。具体的做法是：通过将有理系数 $\frac{k(c-d)}{n}$ 和 $\frac{-kd}{n}$ 四舍五入到最近的整数，得到 Eisenstein 整数商 $(q_1 + q_2\omega)$。对于素域上的 Koblitz 曲线，[25] 中提出了一种使用 Voronoi 单元的方法：原点的 Voronoi 单元由以下直线界定：

$$x + y = \pm 1，\ 2y - x = \pm 1，\ 2x - y = \pm 1。$$

如下图所示：

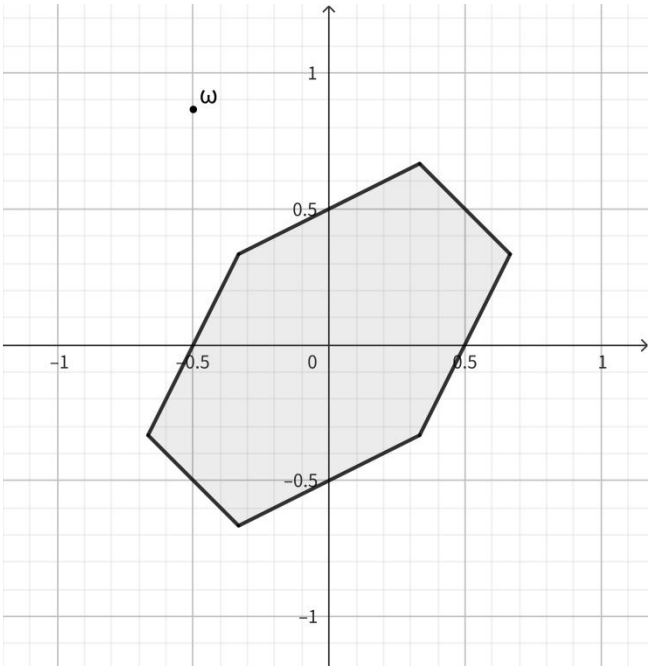

图 3.1 原点的 Voronoi 单元图

该 Voronoi 单元基于范数 N 定义，对于一个点 (x, y) 在 Voronoi 单元 $\mathcal{U}$ 的内部，满足：

$$N(x + y\omega) < N((x + s) + (y + t)\omega)$$

对所有非零整数对 $(s, t) \neq (0,0)$ 成立。也就是说原点的 Voronoi 单元内的点比其他任何格点（Eisenstein 整数）更接近原点。

现在，设 $x = \frac{k(c-d)}{n}$，$y = \frac{-kd}{n}$，[25] 中给出的 q_1, q_2 的精确值为：

$$q_1 = \lfloor \frac{\lfloor x+y \rfloor + \lfloor 2x-y \rfloor + 2}{3} \rfloor, q_2 = \lfloor \frac{\lfloor x+y \rfloor + \lfloor 2y-x \rfloor + 2}{3} \rfloor, \tag{13}$$

最终由 (12) 得到标量分解。这里整数 q_1, q_2 接近 x, y，对于二进制 Koblitz 曲线，文献 [19] 指出在实践中可以简单地直接取分数系数的下整值，而不会影响效率。在我们素数域 Koblitz 曲线的情况下，[9] 中实验表明，采用以下计算方式其性能与使用公式 (13) 相同：

$$q_1 = \lfloor \frac{k(c-d)}{n} \rfloor, q_2 = \lfloor \frac{-kd}{n} \rfloor。 \tag{14}$$

第 4 章 具体实现

4.1 Eisenstein 整数商归约的具体实现

4.1.1 参数定义

在 3.3.1 中使用使用 Lagrange-Gauss 二维格算法构造出来的 (c, d) 为：

$$c = 0xe4437ed6010e88286f547fa90abfe4c3,$$

$$d = -0x3086d221a7d46bcde86c90e49284eb15。$$

4.1.2 分解算法及其预计算

根据上述方法计算出 c, d 后，使我们的任务简化成在 c, d, n 已知的情况下计算 k_1, k_2，我们这里用 (14) 简化版的计算方法，可以采用预计算技巧。在本工作中，我们提出了一种基于整数投影的高精度标量分解方法，用于替代传统浮点投影分解方案。(14) 中，为了将标量 k 分解为满足 (3) $k = k_1 + k_2\lambda \, mod \, n$ 的形式，通常要使用预计算技巧，先根据固定参数 c、d、n 构造两个系数：

$$u = \frac{c-d}{n},\ v = \frac{-d}{n},$$

然后令：

$$q_1 = \lfloor ku \rceil,$$

$$q_2 = \lfloor kv \rceil。$$

最终根据 (12) $k_1 = k - q_1 c + q_2 d$, $k_2 = q_2 d - q_2 c + q_1 d$ 求得分解结果。

但实际实现中，上述计算中的关键问题在于：投影因子 u 和 v 是实数，且往往是无限小数。当标量 k 为 256 位整数时，其与 u 和 v 相乘的结果很难用有限精度的浮点数准确表示。若使用标准 double 类型（最多 53 位有效位），将造成严重的精度损失，进而影响最终分解的正确性。因此，我们采用整数化策略对上述浮点乘法进行改写，从而完全避免浮点运算，同时仍可实现精确的最近整数投影。该方法的核心思想是引入一个足够大的二次幂放大因子 $R = 2^{384}$，将投影系数 u, v 转换为整数比例形式：

$$\tilde{u} = \lfloor \frac{(c-d)\cdot 2^{384}}{n} \rceil,$$

$$\tilde{v} = \lfloor \frac{-d\cdot 2^{384}}{n} \rceil。$$

两者均可在初始化阶段离线预计算并固定存储。在线分解时，我们仅需将 k 与预计算的整数 $\tilde{u}, \tilde{v}$ 相乘，然后右移 384 位（即除以2^{384}），并舍入：

$$q_1 = \lfloor \frac{k \cdot \tilde{u}}{2^{384}} \rfloor,$$

$$q_2 = \lfloor \frac{k \cdot \tilde{v}}{2^{384}} \rfloor。$$

这样一来，我们避免了原始方法中涉及的浮点乘法与浮点取整，所有计算过程均在整数域中完成。由于 2^{384} 的放大足以覆盖 256 位标量与额外精度所需的空间，最终结果的舍入误差仍控制在 ± 0.5 范围内，从而确保所求 q_1, q_2 是最接近目标格点的整数解。此外，乘以 2^{384} 和除以 2^{384} 均可通过移位操作在现代硬件上高效完成，不产生额外运算开销，因此该方法不仅具备高精度保证，还显著提升了运算效率，尤其适合密码库、嵌入式设备等对整数运算资源优化有强烈需求的场景。

同样在 [9] 中对于公式 (13) 未简化版的计算方案也通过相同的预计算加速方法以消除除法运算。在 (13) 中由于 c, d 和 n 是固定的，我们计算 q_1 和 q_2 需要两次整数乘法和两次除法，且除法中涉及的整数较大。因此可以通过一次性预计算（每条曲线一次）简化计算，以便可以移除除法，首先将需要放大的位数表示出来：

$$s = \lceil \log_2 n \rceil, t = \lceil \log_2 2\,(|c| + |d|) \rceil。$$

预计算：

$$\eta_1 = \lfloor \frac{(c-2d)2^{s+t}}{n} \rfloor,$$

$$\eta_2 = \lfloor \frac{(2c-d)2^{s+t}}{n} \rfloor,$$

$$\eta_3 = \lfloor \frac{(-c-d)2^{s+t}}{n} \rfloor。$$

然后对于给定的 $0 \le k < n$，我们计算：

$$f_j = \lfloor \frac{k\eta_j}{2^{s+t}} \rfloor, j = 1,2,3,$$

最后得到 q_1, q_2：

$$q_1 = \lfloor \frac{f_1+f_2+2}{3} \rfloor, q_2 = \lfloor \frac{f_1+f_3+2}{3} \rfloor。$$

可以看出，我们需要计算 3 次乘法 $k\eta_1$，$k\eta_2$，$k\eta_3$，而除以 2^{s+t} 是简单的操作。相比之前的 2 次整数乘法和 2 次除法，计算效率有所提高。

4.1.3 分解结果的界限证明

注意：

（1）在 (12) 中 k_1 和 k_2 的表达式中有相同项 q_2d，可以重复使用。

（2）在求得 q_1 和 q_2 后可以并行计算 k_1 和 k_2，从而达到进一步的加速效果。

下面我们分四个引理证明采用这种计算方式的界限，即证明该函数对于任意输入标量 $k \in [0,n)$ 都能够生成一对足够小的输出 (k_1,k_2)。首先我们定义 $\epsilon_1 = 2^{256}|\frac{\tilde{u}}{2^{384}} - \frac{c-d}{n}|$，$\epsilon_2 = 2^{256}|\frac{\tilde{v}}{2^{384}} - \frac{(-d)}{n}|$。

引理 1 $|q_1 - \frac{k(c-d)}{n}| < \frac{1}{2} + \epsilon_1$.

证明：利用三角不等式：

$$\begin{aligned}|q_1 - \frac{k(c-d)}{n}| &= |q_1 - \frac{k\tilde{u}}{2^{384}} + \frac{k\tilde{u}}{2^{384}} - \frac{k(c-d)}{n}| \\ &\le |q_1 - \frac{k\tilde{u}}{2^{384}}| + k|\frac{\tilde{u}}{2^{384}} - \frac{c-d}{n}| \\ &< \frac{1}{2} + 2^{256}|\frac{\tilde{u}}{2^{384}} - \frac{c-d}{n}| = \frac{1}{2} + \epsilon_1\end{aligned}$$

引理 2 $|q_2 - \frac{k(-d)}{n}| < \frac{1}{2} + \epsilon_2$.

证明：方法同引理 1。

我们知道 (12) $k_1 = k - q_1c + q_2d$, $k_2 = q_2d - q_2c + q_1d$，下边的引理 3 和引理 4 证明 k_1，k_2 的数值边界：

引理 3 $|k_1| < \frac{c+d+1}{2} < 2^{128}$.

证明：根据 (10) $c^2 + d^2 - cd = n$，三角不等式以及引理 1 和引理 2

$$\begin{aligned}|k_1| &= |\frac{k(c^2+d^2-cd)}{n} - q_1c + q_2d| \\ &= |c(\frac{k(c-d)}{n} - q_1) + d(\frac{kd}{n} + q_2)| \\ &\le c|\frac{k(c-d)}{n} - q_1| + d|\frac{kd}{n} + q_2| \\ &< c(\frac{1}{2} + \epsilon_1) + d(\frac{1}{2} + \epsilon_2) \\ &< \frac{(c+d+1)}{2} < 2^{128}\end{aligned}$$

引理 4： $|k_2| < \frac{c}{2} + 1 < 2^{128}$.

证明：根据 三角不等式以及引理 1 和引理 2

$$\begin{aligned}|k_2| &= |k\frac{d(c-d)-d(c-d)}{\mathrm{n}} - q_1 d - q_2(c-d)| \\ &= |d(\frac{k(c-d)}{n} - q_1) + (c-d)(-\frac{kd}{n} - q_2)| \\ &\leq d|\frac{k(c-d)}{n} - q_1| + (c-d)|-\frac{kd}{n} - q_2| \\ &< d(\frac{1}{2} + \epsilon_1) + (c-d)(\frac{1}{2} + \epsilon_2) \\ &< \frac{c}{2} + 1 < 2^{128}\end{aligned}$$

4.2 格上最短向量问题归约的具体实现

4.2.1 参数定义

在 secp256k1 中，满足 (2) $\lambda P = (\beta x, y)$ 的 λ 为：0x5363ad4cc05c30e0a5261c028812645a122e22ea20816678df02967c1b23bd72, 在 3.3.3 中使用扩展欧几里得算法构造格基 $\mathbf{v}_1$ 与 $\mathbf{v}_2$，我们写 $\mathbf{v}_1 = (a_1, b_1)$，$\mathbf{v}_2 = (a_2, b_2)$，有

$$a_1 = 0x3086d221a7d46bcde86c90e49284eb15,$$

$$b_1 = -0xe4437ed6010e88286f547fa90abfe4c3,$$

$$a_2 = 0x114ca50f7a8e2f3f657c1108d9d44cfd8,$$

$$b_2 = 0x3086d221a7d46bcde86c90e49284eb15。$$

4.2.2 分解算法及其预计算

最近格点投影与标量分解过程，是使用基本线性代数在短向量 $\mathbf{v_1}, \mathbf{v_2}$ 生成的整数格中找到一个接近 $(k, 0)$ 的向量 $\mathbf{v}$，然后构造出符合 (3) 的短向量 (k_1, k_2)。参考 [4]《Guide to Elliptic Curve Cryptography》（Algorithm 3.74），使用下述算法：

1. 计算 $\mathrm{c}_1 \approx b_2\mathrm{k}/n$，$\mathrm{c}_2 \approx -b_1\mathrm{k}/n$，其中 $a_1 \cdot b_2 - b_1 \cdot a_2 = n$。

2. 计算 $k_1 = k - (c_1 \cdot a_1 + c_2 \cdot a_2)$，$k_2 = -(c_1 \cdot b_1 + c_2 \cdot b_2)$。 (15)

在实际实现中我们对该算法进行改进，同样整数化策略对上述浮点乘法进行改写，

并在最后一步改变 k_1 的计算方式。在预计算过程中，令：

$$g_1 = \lfloor \frac{2^{384} \cdot b_2}{n} \rfloor,$$

$$g_2 = \lfloor \frac{2^{384} \cdot (-b_1)}{n} \rfloor。$$

对给定标量 k，计算：

$$c_1 = \lfloor \frac{k \cdot g_1}{2^{384}} \rfloor,$$

$$c_2 = \lfloor \frac{k \cdot g_2}{2^{384}} \rfloor。$$

求得：

$$k_2 = -(c_1 \cdot b_1 + c_2 \cdot b_2),$$

$$k_1 = r_1 = k - k_2 \lambda。$$

由于 k 是 256 位整数，这里我们使用 384 位精度以避免精度损失，通过整数乘法与位移来替代高精度除法，提高性能。

4.2.3 分解结果的正确性与界限证明

注意：上边方法中我们 k_1 是通过 $k - k_2 \lambda$ 直接求得，这样可以减少一次乘法运算。下面我们分五个引理证明采用这种计算方式的正确性与界限，前四个引理证明该函数对于任意输入标量 $k \in [0, n)$ 都能够生成一对足够小的输出 (k_1, k_2)。引理 5 证明通过 $k - k_2 \lambda$ 构造出的 k_1（下面的证明里记为 r_1）与引理 3 中的 k_1 等价。

我们定义 $\epsilon_1 = 2^{256} * |\frac{g_1}{2^{384}} - \frac{b_2}{n}|$，$\epsilon_2 = 2^{256} * |\frac{g_2}{2^{384}} - \frac{(-b_1)}{n}|$。

引理 1 $|c_1 - \frac{kb_2}{n}| < \frac{1}{2} + \epsilon_1$.

证明：利用三角不等式：

$$\begin{aligned} |c_1 - \frac{kb_2}{n}| &= |c_1 - \frac{kg_1}{2^{384}} + \frac{kg_1}{2^{384}} - \frac{kb_2}{n}| \\ &\le |c_1 - \frac{kg_1}{2^{384}}| + k|\frac{g_1}{2^{384}} - \frac{b_2}{n}| \\ &< \frac{1}{2} + 2^{256}|\frac{g_1}{2^{384}} - \frac{b_2}{n}| = \frac{1}{2} + \epsilon_1。 \end{aligned}$$

引理 2 $|c_2 - \frac{k(-b_1)}{n}| < \frac{1}{2} + \epsilon_2$.

证明：方法同引理 1。

我们知道 (15) $k_1 = k - (c_1 \cdot a_1 + c_2 \cdot a_2)$，$k_2 = -(c_1 \cdot b_1 + c_2 \cdot b_2)$，下边的引理 3 和引理 4 证明 k_1，k_2 的数值边界：

引理 3 $|k_1| < \frac{a_1+a_2+1}{2} < 2^{128}$.

证明：根据 $a_1 \cdot b_2 - b_1 \cdot a_2 = n$，三角不等式以及引理 1 和引理 2

$$\begin{aligned}
|k_1| &= |\frac{k(a_1 b_2 - b_1 a_2)}{n} - c_1 a_1 - c_2 a_2| \\
&= |a_1(\frac{kb_2}{n} - c_1) + a_2(\frac{k(-b_1)}{n} - c_2)| \\
&\leq a_1|\frac{kb_2}{n} - c_1| + a_2|\frac{k(-b_1)}{n} - c_2| \\
&< a_1(\frac{1}{2} + \epsilon_1) + a_2(\frac{1}{2} + \epsilon_2) \\
&< \frac{(a_1+a_2+1)}{2} < 2^{128}。
\end{aligned}$$

引理 4 $|k_2| < \frac{-b_1+b_2}{2} + 1 < 2^{128}$.

证明：根据 三角不等式以及引理 1 和引理 2

$$\begin{aligned}
|k_2| &= |\frac{k(b_1 b_2 - b_1 b_2)}{\mathrm{n}} - c_1 b_1 - c_2 b_2| \\
&= |b_1(\frac{kb_2}{n} - c_1) + b_2(\frac{k(-b_1)}{n} - c_2)| \\
&\leq (-b_1)|\frac{kb_2}{n} - c_1| + b_2|\frac{k(-b_1)}{n} - c_2| \\
&< (-b_1)(\frac{1}{2} + \epsilon_1) + b_2(\frac{1}{2} + \epsilon_2) \\
&< \frac{(-b_1+b_2)}{2} + 1 < 2^{128}。
\end{aligned}$$

引理 5 $r_1 \equiv k_1 \bmod n$.

证明：由于 $a_1 + b_1\lambda \equiv 0 \bmod n$ 且 $a_2 + b_2\lambda \equiv 0 \bmod n$，

$$\begin{aligned}
r_1 = k - k_2\lambda = k - (-c_1 b_1 - c_2 b_2)\lambda &= k + c_1 b_1 \lambda + c_2 b_2 \lambda \\
&= k - c_1 a_1 - c_2 a_2 = k_1
\end{aligned}$$

4.3 标量乘法其他部分的实现

在标量乘法的具体实现中，基于 Bitcoin Core 社区开发和维护的 libsecp256k1 [26] ，综合使用了第 2 章所述的优化技术：对于固定点加速：针对固定点（如椭圆曲线基点 G），无需使用标量分解，直接采用**预计算表**与**加法链**优化，通过预先生成 16^kG 的倍数点，将实时计算简化为查表累加操作，避免标量分解的额外开销。对于任意点加速：对任意点 P 的标量乘法，在标量分解的基础上，采用**窗口非相邻形式（wNAF）**对指数进行稀疏化编码，将非零比特密度降低至 $\frac{1}{w+1}$，并结合**Shamir's Trick 多重指数算法**计算。除此之外，还参考了批量乘法的多点运算结果，使用 Shamir's Trick 将 $aP + bG$ 类运算合并为协同计算流程，共享中间倍点结果，减少重复运算次数。并采用**雅可比坐标**执行点运算，仅在最终结果处执行一次模逆运算。对于复合运算（如 $aP + bG$），通过保持中间结果的射影坐标形式，延迟逆运算至最终归约阶段，显著降低计算负载。

为了确保标量分解算法的正确性与安全性，在验证函数里实现了对分解函数的输出进行如下两方面的验证：

（1）分解正确性验证：首先验证分解结果是否满足 $k = k_1 + \lambda k_2 \bmod n$，该步骤保证了输入标量 k 被正确分解为两个子标量 k_1, k_2，并满足 GLV 加速方法所要求的代数结构。

（2）验证 k_1, k_2 是否在预期范围内：在 GLV 中我们希望：$|r_2| < B_2$, $|r_2| < B_2$，这两个上界 B_1, B_2 来在第 3 章中证明的分解理论，是理论上最短向量的长度界限，确保加速效果好。

第 5 章 效率分析

本实验在搭载 Apple M4 芯片的 macOS 平台上，使用 CMake 构建，Clang 编译，LLDB 调试，开发环境为 VSCode。我们进行了多组测试，分别测试点乘 GLV 方法平均运算时间和标量分解时间，随机生成的标量为 256 位，每组测试我们都做多次，统计出标量乘法的最长，最短，平均耗时以及标量分解的平均耗时。

首先是固定点 G 的耗时：

表 5-1 固定点 G 标量乘法耗时（单位：微秒μs）

最短	平均	最长	标量分解平均
6.89	6.94	6.96	-

作为对照，我们对不做标量分解的常数乘法测试耗时：

表 5-2 不使用标量分解的标量乘法耗时（单位：微秒μs）

最短	平均	最长	标量分解平均
12.2	12.3	12.3	-

对于任意点 P，采用两种不同归约算法推导出的不同标量分解方法，分别测试它们标量乘法以及标量分解的耗时：

表 5-3 任意点 P 标量分解乘法耗时，采用 4.1 节算法（单位：微秒μs）

最短	平均	最长	标量分解平均
9.80	9.92	9.94	0.0893

表 5-4 任意点 P 标量分解乘法耗时，采用 4.2 节算法（单位：微秒μs）

最短	平均	最长	标量分解平均
10.0	10.2	10.4	0.0908

除此之外，我们还测试了多点运算（批量乘法）的平均耗时，这里采用 4.1 节的标量分解算法，$N = n$ 时表示批量计算：标量₁ $\times$ G + 标量₂ $\times$ P_1 + ... + 标量$_n$ $\times$ P_{n-1}（N 是总点数，包含固定点 G）。

表 5-5 批量乘法耗时（单位：微秒μs）

N	最短	平均	最长	标量分解平均
2	5.85	5.97	6.03	0.0457
3	5.54	5.63	5.68	0.0615
5	5.34	5.38	5.31	0.0743
7	5.22	5.30	5.37	0.0815
11	5.08	5.18	5.27	0.0837
13	5.04	5.14	5.22	0.0861
26	4.96	5.03	5.09	0.0889

从上表中数据我们可以得出如下结论：

（1）对于固定点 G 的乘法最快，虽没有使用标量分解算法，但是通过预计算标量倍数表与高效加法链设计，通过预先生成成 $16^{k}G$ 的形式的结构化数表，将实时计算转化为查表累加操作，能够达到更快的乘法速度。

（2）对于任意点 P，采用两种不同的两种标量分解方法，它们的标量分解速度相差无几，均在 0.09 微秒左右。这是因为虽然推导出两者的归约方法不同，但最后在具体实现中都归约到同样长度的标量运算中去，从代码实现中我们也可以看到两者的实现几乎是对称的。通过两种标量分解实现标量乘法的速度也相差不大，可以印证两种标量分解算法的有效性。

（3）采用标量分解的标量乘法耗时比不采用有显著提升，在我们单点运算的实验中提升约为 24%。

（4）在多点运算的标量乘法中，随着点数增加，平均耗时显著降低，这是因为使用了 Shamir's trick 将多点的乘法合并为一次协同运算，共享中间倍点结果减少了重复运算，平均到单个点上耗时有效下降。在固定点 G 的乘法中不使用标量分解方法，因此随着点数增加平均标量分解时长向 0.09 微秒收敛。

第 6 章 结论

本文围绕素域上 Koblitz 曲线的 GLV 归约方法展开研究，通过理论分析、算法实现与实验验证，系统性地探索了 GLV 方法标量分解问题的归约方法，在标量乘法优化中的应用。以下是本文的主要研究成果与结论：

（1）理论贡献：本文详细分析了 GLV 方法在大特征域 Koblitz 曲线上的适用性，对基于格上向量和基于 Eisenstein 整数环两种标量分解的归约方法，做了更加详细的推导分析，给出了分解结果的上界的证明。

（2）算法实现与优化：本文对基于 Eisenstein 整数归约的标量分解方法给出了具体实现。在 C 语言环境下完成了 secp256k1 曲线的标量分解核心模块，采用了多种优化方法，并创新性地使用了移位整数化策略替代浮点运算，在预计算中避免了高精度除法的性能瓶颈。

（3）实验验证：本文测试了采用不同方法实现标量乘法的平均时间，实验数据表明，采用 GLV 方法的标量乘法表现出显著的性能优势,我们单点运算的实验中提升约为 24%。对于我们实现的新的 GLV 标量分解方法，在个人电脑上标量分解过程平均仅需约 0.09 微秒。

（4）实际应用价值与未来展望：本文标量分解模块的理论推导和实现可为椭圆曲线密码学的主流实现提供优化参考，推动标准曲线的性能改进。secp256k1 曲线是比特币等区块链系统的核心密码学组件，GLV 方法的高效标量分解可显著提升交易签名验证和密钥生成的速度，为高吞吐量区块链网络提供技术支持。通过预计算和移位整数化优化，算法在嵌入式设备或移动终端等计算资源受限的环境中仍能保持高效运行，扩展了椭圆曲线密码学的应用范围。针对未来研究，可以进一步探索曲线点的个数等特殊算数性质在标量分解或是其他标量乘法加速算法上的应用；可以尝试将基于曲线性质的理论归约方法推广到其他具有复乘（CM）性质的曲线上；也可以考虑对现有的归约方法的实现做进一步的效率优化。

参考文献

[1] Koblitz N. CM-curves with good cryptographic properties[C]. Proc. 11th Annu. Int. Cryptol. Conf. Adv. Cryptol, 1992: 279-287.

[2] Koblitz N. An elliptic curves implementation of the finite field digital signature, al-

gorithm[C]. Berlin：Advances in Cryptology-CRYPTO ’98. LNCS 1462 1998:327-337.

[3] Solinas J. Efficient arithmetic on Koblitz curves[J]. Designs, Codes and Cryptography, 2000, 19: 195-249.

[4] Hankerson D., Menezes A., Vanstone S. Guide to elliptic curve cryptography[M]. New York: Springer-Verlag, 2004.

[5] National Institute of Standards and Technology (NIST). Digital signature standard (DSS): DOI: 10.6028/NIST.FIPS[S]. 2019: 186-5-draft.

[6] Gallant R., Lambert R., Vanstone S. Fast point multiplication on elliptic curves with efficient endomorphisms[C]. Crypto 2001. LNCS 2139, 2001: 190-200.

[7] SEC 2: Recommended Elliptic Curve Domain Parameters[EB/OL]. 2010. https://www.secg.org/sec2-v2.pdf.

[8] SP 800-186 Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters[EB/OL].2023. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186.pdf.

[9] Xu G., Han K., Tian Y. On efficient computations of Koblitz curves over prime fields[EB/OL]. Cryptology ePrint Archive, 2024: Paper 2024/1906. [2025-04-26]. https://eprint.iacr.org/2024/1906.

[10] Koblitz N. Elliptic curve cryptosystems[J]. Mathematics of Computation, 1987, 48(177): 203-209.

[11] Miller V. Use of elliptic curves in cryptography[C]. Advances in Cryptology - CRYPTO’85 Proceedings. Lecture Notes in Computer Science 85. LNCS 85. 1985: 417-426.

[12] Elliptic Curve Cryptography -OpenSSLWiki[EB/OL]. [2020-05-02]. https://wiki.openssl.org.

[13] Keylength - NIST Report on Cryptographic Key Length and Cryptoperiod (2019) [EB/OL]. [2020-04-06]. https://www.keylength.com.

[14] Gollmann D., Han Y., Mitchell C. Redundant integer representations and fast exponentiation[J]. Designs, Codes and Cryptography, 1996, 7: 135-151.

[15] How does the MOV attack work?[EB/OL]. Crypto Stack Exchange. https://crypto.stackexchange.com/questions/1871/how-does-the-mov-attack-work.

[16] Cohen H., Miyaji A., Ono T. Efficient elliptic curve exponentiation using mixed coordinates[C].Advances in Cryptology–Asiacrypt ’98.1998: 51-65.

[17] Lim C., Lee P. More flexible exponentiation with precomputation[C].Advances in Cryptology – Crypto ’94.1994: 95-107.

[18] Morain F., Olivos J. Speeding up the computations on an elliptic curve using addition-subtraction chains[J]. Informatique Théorique et Applications, 1990, 24: 531-544.

[19] Silverman J. The arithmetic of elliptic curves[M]. Springer-Verlag, 1986.

[20] Cohen H. A course in computational algebraic number theory[M].3rd printing. Springer-Verlag, 1996.

[21] Babai L. On Lovász' Lattice Reduction and the Nearest Lattice Point Problem[J]. Combinatorica, 1986, 6(1): 1-13.

[22] Rajwade A. R. On rational primes p congruent to 1 (mod 3 or 5)[J]. Proc. Cambridge Philos. Soc., 1969, 66: 61-70.

[23] Wu H., Xu G. On Koblitz curves over prime fields[J]. Journal of Cryptologic Research, 20XX, 0(0): 1-8.

[24] Cohen H. A course in computational algebraic number theory[M].2000.

[25] Brown E., Myers B. T., Solinas J. A. Elliptic curves with compact parameters[R]. Tech. Report, Centre for Applied Cryptographic Research, 2001.

[26] Bitcoin Core Developers. libsecp256k1: Optimized C library for EC operations on curve secp256k1[EB/OL]. GitHub. [2025-04-26]. https://github.com/bitcoin-core/secp-256k1.

致谢

最后，感谢导师许光午教授的悉心指导，感谢本科期间所有的老师、同学，谢谢你们在我的本科学习期间对我的帮助。我还要衷心地感谢父母和朋友，是你们的帮助和支持让我不断成长。

附录

```
/* 标量分解函数 */
static void secp256k1_scalar_split_lambda(secp256k1_scalar *
SECP256K1_RESTRICT r1, secp256k1_scalar * SECP256K1_RESTRICT r2, const
secp256k1_scalar * SECP256K1_RESTRICT k) {
    secp256k1_scalar q1, q2;

    static const secp256k1_scalar minus_d = SECP256K1_SCALAR_CONST(
        0x00000000UL, 0x00000000UL, 0x00000000UL, 0x00000000UL,
        0x3086D221UL, 0xA7D46BCDUL, 0xE86C90E4UL, 0x9284EB15UL
    );
    static const secp256k1_scalar d_minus_c = SECP256K1_SCALAR_CONST(
        0xFFFFFFFFUL, 0xFFFFFFFFUL, 0xFFFFFFFFUL, 0xFFFFFFFEUL,
        0xEB35AF08UL, 0x571D0C09UL, 0xA83EEF72UL, 0x62BB3028UL
    );

    static const secp256k1_scalar u1 = SECP256K1_SCALAR_CONST(
        0xA68B5F3CUL, 0x7098D9F9UL, 0xEB9360F9UL, 0x6885DD3DUL,
        0x6B8A8B71UL, 0x86B2D747UL, 0xE5144715UL, 0x4BB624A2UL
    );
    static const secp256k1_scalar v1 = SECP256K1_SCALAR_CONST(
        0x3A65987AUL, 0xABA3FCDFUL, 0x295D9819UL, 0x2632A0ADUL,
        0xAA37293DUL, 0x08CD7D1AUL, 0x6FE26EB4UL, 0x6F5D632FUL
    );
    SECP256K1_SCALAR_VERIFY(k);
    VERIFY_CHECK(r1 != k);
    VERIFY_CHECK(r2 != k);
    VERIFY_CHECK(r1 != r2);

    /* 移位运算（var）为常数时间 */
    secp256k1_scalar_mul_shift_var(&q1, k, &u1, 384);
    secp256k1_scalar_mul_shift_var(&q2, k, &v1, 384);
    secp256k1_scalar_mul(&q2, &q2, &d_minus_c);
    secp256k1_scalar_mul(&q1, &q1, &minus_d);
    secp256k1_scalar_add(r2, &q1, &q2);
    secp256k1_scalar_mul(r1, r2, &secp256k1_const_lambda);
    secp256k1_scalar_negate(r1, r1);
    secp256k1_scalar_add(r1, r1, k);
    SECP256K1_SCALAR_VERIFY(r1);
    SECP256K1_SCALAR_VERIFY(r2);
#ifdef VERIFY
    secp256k1_scalar_split_lambda_verify(r1, r2, k);
#endif
```

```
}


/* 验证函数 */
static void secp256k1_scalar_split_lambda_verify(const secp256k1_scalar *r1,
const secp256k1_scalar *r2, const secp256k1_scalar *k) {
    secp256k1_scalar s;
    unsigned char buf1[32];
    unsigned char buf2[32];

    static const unsigned char k1_bound[32] = {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,
        0xa2, 0xa8, 0x91, 0x8c, 0xa8, 0x5b, 0xaf, 0xe2, 0x20, 0x16, 0xd0, 0xb9,
0x17, 0xe4, 0xdd, 0x77
    };
    static const unsigned char k2_bound[32] = {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
        0x8a, 0x65, 0x28, 0x7b, 0xd4, 0x71, 0x79, 0xfb, 0x2b, 0xe0, 0x88, 0x46,
0xce, 0xa2, 0x67, 0xed
    };
    secp256k1_scalar_mul(&s, &secp256k1_const_lambda, r2);
    secp256k1_scalar_add(&s, &s, r1);
    VERIFY_CHECK(secp256k1_scalar_eq(&s, k));
    secp256k1_scalar_negate(&s, r1);
    secp256k1_scalar_get_b32(buf1, r1);
    secp256k1_scalar_get_b32(buf2, &s);
    VERIFY_CHECK(secp256k1_memcmp_var(buf1, k1_bound, 32) < 0 ||
secp256k1_memcmp_var(buf2, k1_bound, 32) < 0);
    secp256k1_scalar_negate(&s, r2);
    secp256k1_scalar_get_b32(buf1, r2);
    secp256k1_scalar_get_b32(buf2, &s);
    VERIFY_CHECK(secp256k1_memcmp_var(buf1, k2_bound, 32) < 0 ||
secp256k1_memcmp_var(buf2, k2_bound, 32) < 0);
}
```