

基于椭圆曲线数乘加速的零知识证明高效实现方案

文周之¹⁾ 蒋泽恩¹⁾ 杨浩然¹⁾ 苏 明^{1,2)} 刘晓光¹⁾

¹⁾(南开大学计算机学院、密码与网络空间安全学院 天津 300350)

²⁾(贵州大学公共大数据国家重点实验室 贵阳 550025)

摘 要 零知识证明在区块链等许多领域如区块链扩容、隐私计算有着广泛的应用。生成零知识证明的巨大时间开销限制了其具体的应用。本文关注在生成零知识证明过程中耗时占比约65%的计算多项式承诺步骤,并且大规模椭圆曲线数乘是计算多项式承诺的耗时计算。因此,本文通过引入双基链(DBC)和利用GPU并行计算来加速计算多项式承诺的过程。为了加快椭圆曲线单点数乘速度,本文通过将椭圆曲线数乘计算中的乘数表示为DBC形式、引入椭圆曲线三倍点原子操作等方式降低数乘计算的时间开销。然而,针对大整数的双基链(DBC)计算过程耗费大量时间。为了平衡计算DBC过程和利用DBC计算数乘的时间开销,本文通过计算次优DBC来实现DBC质量和整体计算效率的取舍。具体而言,我们选取多个同一数量级的随机整数,然后对每个整数计算所有可能首项的最短DBC并记录理论时值,即通过该双基链计算得到的标量乘法的理论时间。在计算次优DBC时,我们仅考虑那些期望理论时值最小的若干首项,这使得计算次优DBC相较于计算最优DBC效率更高,而DBC的质量仅有小幅下降。针对计算多项式承诺时需要计算的椭圆曲线大规模数乘,本文通过CUDA在GPU上实现了椭圆曲线计算库,利用SIMD模式为多项式中的每一项分配单独线程,具体来说,我们将椭圆曲线点乘任务分配给每个线程,每个线程通过DBC方式计算出多项式承诺的结果。实验表明,对于256位椭圆曲线,针对单次椭圆曲线数乘,使用本文提出的次优DBC算法计算平均情况下比OpenSSL的实现快8.2%。同时,对于不同大小的大整数标量,使用次优DBC完成椭圆曲线点乘均有显著的速度提升。且在最坏情况下,DBC算法的计算速度也不会低于NAF椭圆曲线点乘算法。在Nvidia 4090 GPU上,相比于libsark的多项式承诺实现,本文提出的GPU实现方案在处理较大规模(超过 2^{15} 个点)的椭圆曲线点乘时,相较于同级别的CPU计算实现了5.76倍的加速。相当于把常用libsark生成零知识证明的速度加倍。同时,对比基于Pippenger算法设计的椭圆曲线数乘方案,本方案在处理稀疏椭圆曲线多项式的情形有着明显优势。对于 d 次椭圆曲线多项式,其包含的非零项越少,本方案相比于与基于Pippenger算法的椭圆曲线点乘算法的速度提升就越明显。例如,对于非零项数量为 $\lceil \sqrt{d} \rceil$ 的 d 次的稀疏椭圆曲线多项式,当 d 的取值范围为 2^{18} 至 2^{20} 时,我们的方案相较于基于Pippenger算法的椭圆曲线点乘算法有约1.8倍的速度提升。因此我们的方案在涉及大量稀疏椭圆曲线多项式计算的场景中具有广泛的应用前景。

关键词 零知识证明;区块链;双基数字系统;椭圆曲线密码学;GPU计算

中图分类号 TP309

DOI号 10.11897/SP.J.1016.2025.02967

An Efficient Implementation of Zero-knowledge Proof Based on Acceleration of Scalar Multiplication

WEN Zhou-Zhi¹⁾ JIANG Ze-En¹⁾ YANG Hao-Ran¹⁾ SU Ming^{1,2)} LIU Xiao-Guang¹⁾

¹⁾(College of Computer Science & College of Cryptology and Cyber Science, Nankai University, Tianjin 300350)

²⁾(State Key Laboratory of Public Big Data, Guizhou University, Guiyang 550025)

Abstract Zero-knowledge proof has wide applications in blockchain and many areas, such as

收稿日期:2024-12-03;在线发布日期:2025-07-28。本课题得到公共大数据国家重点实验室开放课题(PBD2022-12)、天津市科技计划重点研发计划(19YFZCSF00900, 20JCZDJC00610)、国家自然科学基金项目(62272253)资助。文周之,硕士,主要研究方向为密码学、区块链技术、零知识证明等。Email:occulicplus@outlook.com。蒋泽恩,硕士,主要研究方向为区块链隐蔽信道。杨浩然,硕士,主要研究方向为零知识证明、全同态加密。苏 明(通信作者),博士,副研究员,中国计算机学会(CCF)会员,主要研究领域包括序列复杂度和相关算法、数字水印、区块链等。Email:suming@nankai.edu.cn。刘晓光,博士,教授,主要研究方向为搜索引擎、存储系统和GPU计算等。

scaling for blockchain and privacy computing. But generating zero-knowledge proofs costs substantial time, which restricts their practical applications. The step of polynomial commitment computation accounts for approximately 65% of the time in generating zero-knowledge proofs, and the time-consuming computation of polynomial commitments is a large number of elliptic curve scalar multiplication. So, we accelerate the scalar multiplication by double-base chains (DBC) and GPU parallel computing. In order to accelerate the speed of a single scalar multiplication, we represent the scalar as DBC form and add triple point atomic operation of elliptic curves to reduce the time cost of scalar multiplication. However, the DBC computation process for large integers cost a lot of time. To balance the time cost between the DBC computation process and using DBC for scalar multiplication, we achieve a trade-off between DBC quality and overall computational efficiency through suboptimal DBCs. Specifically, we select multiple random integers of the same order of magnitude, then we calculate the first term of the shortest DBC with each possible leading term for each integer and record the theoretical time value, i. e. , the theoretical time of scalar multiplication calculated by this DBC. When calculating the suboptimal DBC, we only consider the several leading terms with the smallest expected theoretical time value, which enables the suboptimal DBC to be more efficient compared to the optimal DBC, yet with a small decrease in DBC quality. For the large-scale scalar multiplication in computing polynomial commitments, we implement an elliptic curve computation library on the GPU by CUDA with assigning individual threads to each term in the polynomial in SIMD mode. Specifically, we assign the tasks of scalar multiplication to each thread, and each thread computes the results of polynomial commitments via the DBC method in parallel. Experiments demonstrates that for 256-bit elliptic curves, our suboptimal DBC algorithm for computing single scalar multiplication is 8.2% faster than the OpenSSL implementation. Meanwhile, for scalars of different sizes, the suboptimal DBC algorithm achieves a significant speedup in scalar multiplication. And in the worst case, the DBC algorithm is better than the typical NAF scalar multiplication. For the computation of polynomial commitment, our GPU implementation scheme proposed on an Nvidia 4090 GPU achieves a 5.76x speedup for large-scale scalar multiplication (over 2^{15} points) compared to the libsnark running on a CPU of the same level, which is equivalent to doubling the speed of generating zero-knowledge proofs by libsnark. Meanwhile, compared with the scalar multiplication based on the Pippenger algorithm, our scheme has significant advantages in handling sparse elliptic curve polynomials. For a degree- d elliptic curve polynomial, the fewer non-zero terms it contains, the more significant the speedup of our scheme becomes compared to scalar multiplication based on the Pippenger algorithm. For example, for a sparse elliptic curve polynomial of degree d having $\lceil \sqrt{d} \rceil$ nonzero terms, our scheme achieves about 1.8 speedup compared with the scalar multiplication based on the Pippenger algorithm when $d = 2^{18}$ to 2^{20} . Therefore, our scheme has promising applications involving scalar multiplications for a large number of sparse elliptic curve polynomials.

Keywords zero-knowledge proof; blockchain; double base number system; elliptical curve cryptography; GPU computing

1 引 言

自从 Goldwasser 等人提出零知识证明以来^[1],

这项技术就受到了众多研究者的广泛关注。近年来,零知识证明被应用于多个领域,其相关工作在2012年获得了图灵奖。零知识证明的核心在于,证明者能够在不向验证者泄露任何隐私信息的情况

下,使验证者相信某个论断是正确的,这使得零知识证明在隐私保护的场景下得到了广泛应用。例如金融领域中为了保护金额隐私的范围证明(如Bulletproofs^[2]和Springproofs^[3])、保护交易隐私的数字货币Zcash^[4]和Monero^[5]、zk-rollup^[6]、安全多方计算等等。其中zk-rollup给出了在区块链系统中,将密集计算任务在链下进行,使用零知识证明对数据进行压缩上链^[6]从而达到扩容的需求的方案。为使零知识证明可应用于通用计算,通用的非交互式零知识证明方案在近年来获得了广泛的研究。在这其中,zk-SNARK (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge, 简洁非交互式零知识证明)是目前主流的通用型零知识证明协议之一。zk-SNARK能够为特定问题生成简洁的计算证明,且生成的证明验证成本较低,所以其得到了广泛应用^[4,7],比如可验证机器学习^[8]通过将零知识证明引入机器学习模型的训练过程,在保护数据和算法模型的同时验证某次输出的结果准确可信。其他相关工作还有利用区块链和零知识证明保护隐私的区块链彩票方案^[9],可验证数据外包^[10],匿名投票^[11]等。

已有的zk-SNARK算法通常对证明的验证速度很快,但是生成证明的速度较慢,这成为了协议性能的瓶颈^[12]。zk-SNARK将特定问题转化为电路约束问题,在实际应用中可达数十万个电路约束。例如,在Plonk协议中,问题被转化为计算电路后,电路的约束被分为门约束和复制约束。用一元多项式表示门约束,通过计算所有多项式聚合后的结果,证明者可以给出一个多项式承诺(Polynomial Commitment)来证明其拥有一个秘密输入值满足所有约束。证明者通过输入约束条件生成证明信息,而该过程中存在大量有限域上的椭圆曲线数乘运算(即计算 $\sum n_i \times \pi_i$, n_i 为标量, π_i 为椭圆曲线点)。椭圆曲线密码学(Elliptic Curve Cryptography,下文简称ECC)是现代密码学的一项基础理论。虽然椭圆曲线公钥仅需要较低的位数就可以达到传统RSA公钥的安全性(256bit的ECDSA安全性与2048bit的RSA安全性相当),但是计算或验证一个椭圆曲线签名(需计算椭圆曲线数乘)的时间开销却非常大,这影响了实际密码学应用实现的性能,也使得类似区块链的密码学应用遭遇计算性能瓶颈。

在实际的zk-SNARK应用中,多项式的次数直接受到由隐私保护计算问题转换而成的电路复杂度的影响,而对密码学领域内常见的算法而言,其构建的电路往往复杂度较高。表1列出了构建不同类型

电路所需的具体约束数量,从中可以看出,构建Zcash验证交易所需超过 2×10^6 个电路约束。

表1 Groth16协议为密码学原语构建电路中的约束个数

问题类型(规模256比特)	电路中约束个数
SHA256 ^[13]	27 280
3阶Merkle树 ^[14]	84 150
4阶Merkle树	112 198
5阶Merkle树	140 246
Zcash验证交易 ^[4]	大于 2×10^6

在计算其相应的多项式承诺时,需要将这些约束转化为椭圆曲线点乘,因此需要进行大规模的椭圆曲线点乘运算(MSM, Multi-Scalar Multiplication),这成为zk-SNARK的性能瓶颈。图1表明整个椭圆曲线点乘计算的时间占生成证明时间高达65%。椭圆曲线点乘计算是zk-SNARK的主要性能瓶颈,仅此一项即可占据65%的证明生成时间(图1)。为此,研究如何优化大规模点乘及多项式承诺的生成与验证,对于提升整体性能具有重要的理论价值与应用前景。本文聚焦于加速计算多项式承诺计算这一主题,从软件实现的多个层级优化多项式承诺计算过程。

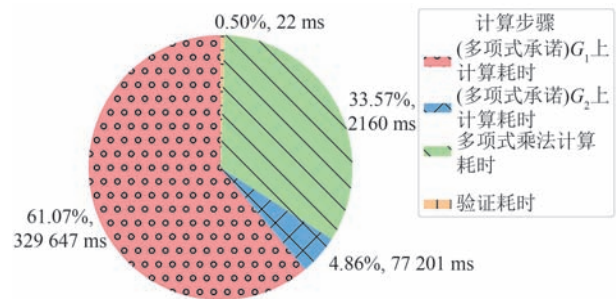


图1 zk-SNARK各步骤耗时比例

在椭圆曲线数乘方面,本文基于双基数字表示系统(Double Base Number System, DBNS)设计了双基链(Double Base Chain, DBC),以此得到可以通过更少的有限域上基本运算完成椭圆曲线点乘的算法。针对zk-SNARK中的大规模椭圆曲线数乘计算,本文采用GPU并行技术进行加速,通过对有限域上椭圆曲线基本运算的分析,本文实现了高效的大规模椭圆曲线数乘计算方法。实验结果表明,基于DBC的单点椭圆曲线计算速度比于通用的OpenSSL库的椭圆曲线计算快约8.2%,对于大规模的椭圆曲线数乘计算,本文在Nvidia 4090 GPU上实现的并行计算性能可达libsnark库的5.76倍,这相当于将libsnark生成证明的速度加倍。

2 背景知识和相关工作

2.1 符号标记解释说明

本文中出现的符号含义如表2所示:

符号标记	说明
$\mathbb{F}(\rho)$	模素数 ρ 的有限域
\mathbb{G}	椭圆曲线有限群
g	椭圆曲线群上的生成元
π	随机椭圆曲线点
$[\chi]_1$	\mathbb{G}_1 有限群上的元 χ
$[\psi]_2$	\mathbb{G}_2 有限群上的元 ψ
$e(g_1, g_2)$	双线性配对
η_1	标量, 椭圆曲线数乘操作的乘数
a, b	椭圆曲线 Weierstrass 形式 $y^2=x^3+ax+b$ 中的参数
ρ	椭圆曲线计算模数
$U(x), V(x), W(x), \Pi(x)$	关于 x 的多项式
$[U]_j(x), [V]_j(x), [W]_j(x), [\Pi]_j(x)$	\mathbb{G}_j 有限群上关于 x 的椭圆曲线多项式($j=1, 2, \dots$)
M	\mathbb{F} 上整数乘法耗时单位
S	\mathbb{F} 上整数平方耗时单位
$*$	椭圆曲线数乘
$+$	有限域元素之间的加法
\oplus	椭圆曲线元素之间的加法

2.2 Groth16协议

Groth16协议^[12]作为目前zk-SNARK的一种代



图2 Groth16生成证明过程

$$[\sigma_1]_1 = \left(q, r, \left\{ x^i \right\}_{i=0}^{m_{tot}-1}, \left\{ \frac{rU_i(x) + qV_i(x) + W_i(x)}{o} \right\}_{i=0}^{m_{tot}-1}, \left\{ \frac{rU_i(x) + qV_i(x) + W_i(x)}{t} \right\}_{i=l+1}^{m_{tot}-1}, \left\{ \frac{x^i R(x)}{t} \right\}_{i=0}^{m_{tot}-2} \right) * g_1, [\sigma_2]_2 = \left(r, o, t, \left\{ x^i \right\}_{i=0}^{m_{tot}-1} \right) * g_2.$$

(2) 生成证明: Prover 随机选择两个参数 $u, v \in \mathbb{Z}_\rho$, 根据 CRS 和见证输入, 构造线性组合, 计算多项式承诺(Polynomial Commitment): 即生成证明 $\pi([\chi]_1, [\psi]_2, [\omega]_1)$:

表性方案, 已经在多个场景中^[4, 13-15]有所应用。Groth16生成证明的方式如图2所示。

如图2所示, Groth16算法首先将待计算程序转换为R1CS(Rank-1 Constraint System)电路, 之后将该电路的可满足问题转换为QAP问题(Quadratic Arithmetic Program)^[16], 即证明者在不公开证据的情况下, 构造目标多项式 $R(x)$ 使得其能够多项式整除QAP多项式。换言之, 需要找到一多项式 $H(x)$ 满足:

$$H(x) = \frac{\left(\sum_{i=0}^{m_{deg}} s_i W_i(x) - \sum_{i=0}^{m_{deg}} s_i U_i(x) \cdot \sum_{i=0}^{m_{deg}} s_i V_i(x) \right)}{R(x)}$$

其中 $\sum_{i=0}^{m_{deg}} s_i W_i(x) - \sum_{i=0}^{m_{deg}} s_i U_i(x) \cdot \sum_{i=0}^{m_{deg}} s_i V_i(x)$ 为QAP多项式, s_i 是证明生成过程中构造的一系列见证输入, m_{deg} 表示多项式的项数, 等于公共输入、私密输入与中间变量总数的总和, $U(x), V(x), W(x)$ 为通过插值方法得到的多项式。验证者通过重构这些多项式之间的整除关系来验证证明的正确性。在Groth16协议中, 具体的计算过程主要分为三步(m_{tot} 表示电路输入的总个数, m_{com} 表示公共输入的总个数):

(1) 建立阶段: 可信第三方通过随机选取 $q, r, o, t, x \in \mathbb{Z}_\rho^*$ (对证明者和验证者不公开), 计算公共参数CRS:

$$[\chi]_1 = \left(q + ut + \sum_{i=0}^{m_{deg}} s_i U_i(x) \right) * g_1,$$

$$[\psi]_2 = \left(r + vt + \sum_{i=0}^{m_{deg}} s_i V_i(x) \right) * g_2,$$

$$[\omega]_1 = \left(\frac{\sum_{i=m_{com}+1}^{m_{deg}} s_i(qU_i(x) + rV_i(x) + W_i(x)) + H(x)R(x)}{t} - wvt \right) * g_1 \oplus v * [\chi]_1 \oplus u * [\psi]_1. \quad (1)$$

(3)验证证明: Verifier通过椭圆曲线双线性配对检查多项式之间的关系是否成立:

$$e([\chi]_1, [\psi]_2) = e(q * g_1, r * g_2) \oplus e([\omega]_1, t * g_2) \oplus e\left(\frac{\sum_{i=0}^{m_{com}} rU_i(x) + qV_i(x) + W_i(x)}{o} * g_1, o * g_2\right).$$

如果等式成立,则验证通过,否则拒绝。由于证明者和验证者不知道 x 的输入,因此需要通过CRS中的 $\{x^i \times g\}$ 数组来计算多项式承诺,而计算多项式承诺相当于计算一系列的椭圆曲线多项式。具体地,对于一般的多项式 $\Pi(x)$:

$$\Pi(x) = c_0 + c_1x + c_2x^2 + c_3x^3 + \dots \quad (2)$$

其在群 G_1 上的椭圆曲线多项式形如:

$$[\Pi]_1(x) = c_0 \times g_1 \oplus c_1 \times (x \times g_1) \oplus c_2 \times (x^2 \times g_1) \oplus c_3 \times (x^3 \times g_1) \oplus \dots \quad (3)$$

这意味着实际计算 $U(x), V(x), W(x)$ 时需要计算的是椭圆曲线多项式 $[U]_1(x), [V]_2(x), [W]_1(x)$ 。

2.3 椭圆曲线数乘

椭圆曲线可以通过定义在大素数域 \mathbb{F}_p 上的Weierstrass形式来描述,即 $y^2 = x^3 + ax + b$,其中 $a, b \in \mathbb{F}_p, 4a^3 + 27b^2 \neq 0$ 。之后可在满足此方程的离散点上定义加法操作,构造椭圆曲线加法循环群。为了更好地处理无穷远点,同时减少 \mathbb{F}_p 上的乘法逆元操作,在算法实现层面常用Jacobi射影坐标 (X, Y, Z) 表示椭圆曲线点。对于非无穷远点,射影坐标和仿射坐标之间的转化关系为 $x = X/Z^2, y = Y/Z^3$,无穷远点在射影坐标中可用 $(0, 1, 0)$ 表示。作为区分,本文使用 \oplus 表示椭圆曲线点加, $+$ 表示 \mathbb{F}_p 上的加法。椭圆曲线数乘(scalar multiplication)形式为 $\pi_{ans} = n \times \pi$,其中乘数 n 为 \mathbb{F}_p 上的整数, π 为椭圆曲线点。计算椭圆曲线数乘通常依赖于椭圆曲线上的点加和二倍点计算,这些计算的实际耗时可以由 \mathbb{F}_p 上的乘法(耗时单位记为 M)与平方(耗时单位记为 S)计算的次数来刻画。例如,对于 $a = -3$ 的椭圆曲线,在Jacobi射影坐标形式下计算点加和二倍点理论耗时分别为 $11M + 5S$ 和 $3M + 5S$ ^[17]。几十年来已经有许多学者提出了不同的算法来改进数乘运算的效率^[18-22],其中主要的思想是使用不同

的形式表示乘数 n ,包括普通二进制表示、NAF(Non-Adjacent Format,非相邻形式)、w-NAF以及DBNS(Double Base Number System,双基数字系统)来减少二倍点和点加的次数,以 $n=314\ 159$ 为例,不同形式的表示如下:

binary表示: $314\ 159 = 2^{18} + 2^{15} + 2^{14} + 2^{11} + 2^9 + 2^8 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0$.

NAF表示: $314\ 159 = 2^{18} + 2^{16} - 2^{14} + 2^{12} - 2^{10} - 2^8 + 2^6 - 2^4 - 1$.

4-NAF表示: $314\ 159 = 5 \cdot 2^{16} - 3 \cdot 2^{12} - 5 \cdot 2^8 + 3 \cdot 2^4 - 1$.

DBNS表示: $314\ 159 = 2^{12}3^6 + 2^{10}3^5 - 2^83^3 + 2^53^1 - 2^33^0$.

不同的表示会影响数乘实现的效率。目前已有的较好结果是Vassil Dimitrov^[23]等人提出的双基链(Double Base Chain, DBC)形式。这里的DBC是一种特殊的DBNS,即使用 $w_n = \sum_{i=1}^l \gamma_i 2^{\alpha_i} 3^{\beta_i}$ 来表示数字 n 的一种方式,其中 $c_i \in \{\pm 1\}, \alpha_i, \beta_i$ 满足 $0 \leq \alpha_1 \leq \alpha_2 \leq \dots \leq \alpha_l, 0 \leq \beta_1 \leq \beta_2 \leq \dots \leq \beta_l$ 。2020年,Wei等人通过动态规划算法,给出了一个可以在多项式时间内计算最优DBC的方案^[22]。2021年,Eom等人分析了Wei Yu使用动态规划的DBC算法,并在此基础上提供了在小幅牺牲DBC质量的情况下大幅提高DBC计算速度的算法^[24]。本文在Eom等人的算法基础上进行了改进,得到在实际零知识证明应用中,整体计算时间更小的DBC计算方案。

2.4 GPU平台计算

鉴于生成zk-SNARK证明速度较慢,目前已有研究工作针对zk-SNARK生成证明加速展开研究。一部分研究希望通过硬件设计加速zk-SNARK生成证明的速度,例如FPGA硬件加速方案和流水线计算架构^[25-27],但这些工作依赖于特定的芯片架构,缺乏泛用性更强的加速方案。而GPU是目前应用最广泛、对通用计算支持性最好的异构计算平台之一。得益于GPU作为图形核心的强大并行能力,只要计算数据之间具有弱依赖性,多线程的任务计算方案就可以利用GPU实现。针对传统密码学应用,已有一部分GPU平台上的密码学计算库的设计和实现,如加密算法的GPU并行计算^[28]。目前,有人探索使用GPU加速zk-SNARK的计算速度,

例如Ni等人^[29]通过加速多项式乘法NTT操作实现加快证明的生成速度, Lu等人^[30]通过椭圆曲线点乘加速和多项式NTT加速, 在GPU加速的加密货币应用程序Filecoin上实现了2.18倍的加速。Ma等人^[31]利用细粒度的任务并行策略, 在V100显卡上实现了17.6倍-48.1倍的生成证明加速。考虑到Groth16生成证明步骤需要计算大规模椭圆曲线数乘, 这部分内容各椭圆曲线点数乘计算相对独立, 便于使用GPU进行并行计算。因此本文从研究椭圆曲线点乘加速出发, 设计加速zk-SNARK零知识证明的方案。具体来说, 本文将椭圆曲线双基链(Double Base Chain, DBC)融合进椭圆曲线倍点的计算中, 结合椭圆曲线倍点的DBC加速, 相比于libsnark库, 在椭圆曲线数乘步骤, 本方案可以在Nvidia 4090 GPU上实现5.76倍的加速效果。

图3给出了本方案的计算架构。

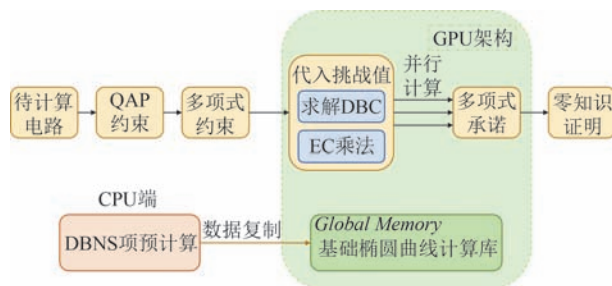


图3 计算架构

3 使用DBC计算椭圆曲线数乘

3.1 DBC简介

2020年, Wei等人^[22]给出了使用双基链(Double-Base Chain, DBC)完成椭圆曲线数乘操作的算法, 该算法在使用DBC计算椭圆曲线的步骤, 相较于传统的使用NAF算法完成数乘操作有着13%的加速比, 但该方案只考虑使用DBC计算椭圆曲线数乘时的加速, 而忽略了计算最优DBC本身的时间。因此本文在将DBC方案融入椭圆曲线数乘加速方案当中的同时, 给出权衡使用DBC计算椭圆曲线数乘和计算DBC时间的次优DBC算法。双基链是一种特殊的双基数字系统(Double-Base Number System), 而双基数字系统是使用 $\sum_{i=1}^l \gamma_i 2^{\alpha_i} 3^{\beta_i}$ 表示数字 n 的一种方式。若 $n = \sum_{i=1}^l \gamma_i 2^{\alpha_i} 3^{\beta_i}$, 其中 $\gamma_i \in \{-1, 1\}$ 。如果 α_i, β_i 满足 $0 \leq \alpha_1 \leq \alpha_2 \leq \dots \leq \alpha_l, 0 \leq \beta_1 \leq \beta_2 \leq \dots \leq \beta_l$, 那么

$\sum_{i=1}^l \gamma_i 2^{\alpha_i} 3^{\beta_i}$ 这一表示称为正整数 n 的DBC表示, 其中, 绝对值最大的项即 $2^{\alpha} 3^{\beta}$ 称为DBC的首项。本文用 w_n 来表示 n 的DBC表示。

为了更好地衡量DBC的质量, 本文给出了汉明重量和理论时值两个概念。考虑到计算椭圆曲线倍点的理论时值主要由二倍点、三倍点、点加三种操作的次数决定。记点加的平均计算时间为 A , 二倍点的平均计算时间为 D , 三倍点的平均计算时间为 T , 如果 w_n 为 $\sum_{i=1}^l \gamma_i 2^{\alpha_i} 3^{\beta_i}$, 那么该DBC的汉明重量为 l , 理论时值为 $E(w_n) = (l-1) \cdot A + \alpha_l \cdot D + \beta_l \cdot T$ 。在此基础上, 定义具有最小理论时值的DBC为最优DBC(Optimal Double-Base Chain)^[22]; 对于正整数 n 的首项为 $2^{\alpha} 3^{\beta}$ 的所有DBC表示而言, 记其中汉明重量最小的DBC表示为(首项为 $2^{\alpha} 3^{\beta}$ 的)最短DBC(Shortest Double-Base Chain)。

作为DBC表示的样例, 表2给出了正整数 n_{example} 在几个不同首项的情况下计算出次优DBC的理论时值, 其中 $n_{\text{example}} = (6B17DDF0B09E408F80F81DBEDC76F0DD0EF6471A11BD0911FECC5297751005F4)_{16}$ 为本文随机生成的一个256位长数字。从表3可以看出, 对于正整数 n_{example} 的DBC表示而言, 针对选定的5个DBC首项, 按照 $S = 0.8 \text{ M}$ 计算, 首项为 $2^{139} 3^{73}$ 的次优DBC的理论时值最小。

表3 不同首项的次优DBC的理论时值

DBC首项	汉明重量	次优DBC的理论时值
$2^{193} 3^{39}$	71	2892.4 M
$2^{166} 3^{56}$	59	2737.6 M
$2^{139} 3^{73}$	49	2612.8 M
$2^{112} 3^{90}$	60	2878.0 M
$2^{90} 3^{104}$	75	3050.4 M

当给定的正整数 n 的DBC表示形式 $w_n = \sum_{i=1}^l \gamma_i 2^{\alpha_i} 3^{\beta_i}$, 可按照算法1所示的方式利用DBC完成椭圆曲线点乘计算。

算法1. 基于DBC的椭圆曲线点乘^[23]

输入: 整数 n 的DBC表示形式 $w_n = \sum_{i=1}^l \gamma_i 2^{\alpha_i} 3^{\beta_i}$ ($\gamma_i = \pm 1$), 椭圆曲线点 a

输出: 点乘结果 r

- $\bar{a} \leftarrow$ 计算点 a 的逆元
- $r \leftarrow a$
- $now_dbl \leftarrow \alpha_l, now_tpl \leftarrow \beta_l$
- FOR $i = l - 1$ downto 0 DO
- WHILE $now_dbl > \alpha_i$ DO
- $r \leftarrow 2 * r$

```

7.   now_dbl = now_dbl - 1
8.   END WHILE
9.   WHILE now_tpl > βi DO
10.    r ← 3 * r
11.    now_tpl = now_tpl - 1
12.  END WHILE
13.  IF γi == -1 THEN
14.    r ← r ⊕ a
15.  ELSE
16.    r ← r ⊕ a
17.  END IF
18. END FOR
19. WHILE now_dbl > 0 DO
20.   r ← 2 * r
21.   now_dbl = now_dbl - 1
22. END WHILE
23. WHILE now_tpl > 0 DO
24.   r ← 3 * r
25.   now_tpl = now_tpl - 1
26. END WHILE
27. RETURN r

```

3.2 最优DBC的求解

Wei Yu等人^[22]通过定义子链,给出了从 n 的低位算起再向高位逐渐扩展的方式计算最优DBC的算法。如果 n 的DBC表示为 $\sum_{i=1}^l \gamma_i 2^{\alpha_i} 3^{\beta_i}$,那么当 $l_0 \leq l$ 时, $\sum_{i=1}^{l_0} \gamma_i 2^{\alpha_i} 3^{\beta_i}$ 为 w_n 的子链形式。如果子链形式的首项系数 $\gamma_{l_0} = 1$,那么该子链形式为正链,反之该子链形式为负链。

对于给定的 n ,本文定义 $n_{\alpha,\beta} = n \bmod 2^{\alpha} 3^{\beta}$ 。同时定义 $w_n(\alpha, \beta)$ 和 $\bar{w}_n(\alpha, \beta)$ 用于表示 $n_{\alpha,\beta}$ 的DBC, $w_n(\alpha, \beta)$ 为正链, $\bar{w}_n(\alpha, \beta)$ 为负链。根据 $n_{\alpha,\beta}$ 的定义可知,必存在整数 k 满足 $n = n_{\alpha,\beta} + 2^{\alpha} 3^{\beta} \cdot k$,因此 $w_n = w_n(\alpha, \beta) + 2^{\alpha} 3^{\beta} \cdot w_k$,显然这是 n 的一个DBC表示。由此可知 $w_n(\alpha, \beta)$ 和 $\bar{w}_n(\alpha, \beta)$ 均为 w_n 的子链。

在此基础上,预定义 $w_n(0, 0) = 0, \bar{w}_n(0, 0) = \text{NULL}$ 。为方便计算,定义 $w_n(i, -1) = \text{NULL}, \bar{w}_n(i, -1) = \text{NULL}, w_n(-1, j) = \text{NULL}, \bar{w}_n(-1, j) = \text{NULL}$ 。同时定义NULL和任何数相加都是NULL。

当 $d_{\alpha,\beta} = \left\lfloor \frac{n_{\alpha,\beta}}{2^{\alpha-1} 3^{\beta-1}} \right\rfloor$ 取值给定,可以推出描述 $w_n(\alpha, \beta)$ 和 $\bar{w}_n(\alpha, \beta)$ 与 $w_n(\alpha-1, \beta), w_n(\alpha, \beta-1), \bar{w}_n(\alpha-1, \beta), \bar{w}_n(\alpha, \beta-1)$ 的递推关系的引理^[22]:

引理1. 记 $d_{\alpha,\beta} = \left\lfloor \frac{n_{\alpha,\beta}}{2^{\alpha-1} 3^{\beta-1}} \right\rfloor$,则有如下结论。

(1)若 $d_{\alpha,\beta} = 0$ 或1,则

$$w_n(\alpha, \beta) = \min_L \{ w_n(\alpha-1, \beta), w_n(\alpha, \beta-1), 2^{\alpha} 3^{\beta-1} + \bar{w}_n(\alpha, \beta-1) \},$$

$$\bar{w}_n(\alpha, \beta) = -2^{\alpha-1} 3^{\beta} + \bar{w}_n(\alpha-1, \beta).$$

(2)若 $d_{\alpha,\beta} = 2$,则

$$w_n(\alpha, \beta) = \min_L \{ w_n(\alpha-1, \beta), 2^{\alpha-1} 3^{\beta} + \bar{w}_n(\alpha-1, \beta), 2^{\alpha} 3^{\beta-1} + w_n(\alpha, \beta-1) \}.$$

$$\bar{w}_n(\alpha, \beta) = -2^{\alpha-1} 3^{\beta} + \bar{w}_n(\alpha-1, \beta).$$

(3)若 $d_{\alpha,\beta} = 3$,则

$$w_n(\alpha, \beta) = 2^{\alpha-1} 3^{\beta} + w_n(\alpha-1, \beta),$$

$$\bar{w}_n(\alpha, \beta) = \min_L \{ -2^{\alpha-1} 3^{\beta} + w_n(\alpha-1, \beta), \bar{w}_n(\alpha-1, \beta), -2^{\alpha} 3^{\beta-1} + \bar{w}_n(\alpha, \beta-1) \}.$$

(4)若 $d_{\alpha,\beta} = 4$ 或5,则

$$w_n(\alpha, \beta) = 2^{\alpha-1} 3^{\beta} + w_n(\alpha-1, \beta),$$

$$\bar{w}_n(\alpha, \beta) = \min_L \{ \bar{w}_n(\alpha-1, \beta), -2^{\alpha-1} 3^{\beta} + w_n(\alpha, \beta-1), \bar{w}_n(\alpha, \beta-1) \}.$$

其中, $\min_L \{ w_a, w_b \}$ 返回 w_a, w_b 中汉明重量最小的非NULL双基链,如果 w_a 和 w_b 均为NULL,则返回NULL, $\min_L \{ w_a, w_b, w_c \}$ 同理。本引理完整的证明过程见文献[22]。

基于引理1,本文可以得到求解最优DBC的递推算法2,其中 $\min_V \{ w_a, w_b \}$ 返回 w_a, w_b 中理论时值较小的双基链,NAF(n)表示获取正整数 n 的NAF表示。

算法2. 最优DBC求解的递推算法

输入:正整数 n

输出: n 的最优DBC

```

1.   wn(0, 0) ← 0, w̄n(0, 0) ← NULL, wmin ← NAF(n)
2.   FOR α = 0: ⌊log2 2n⌋ DO
3.     wn(α, -1) ← NULL
4.     w̄n(α, -1) ← NULL
5.   END FOR
6.   FOR β = 0: ⌊log3 2n⌋ DO
7.     wn(-1, β) ← NULL
8.     w̄n(-1, β) ← NULL
9.     Bound[β] ← log3  $\frac{B}{3^{\beta}}$ 
10.  END FOR

```

```

11. FOR  $\beta=0:\lfloor \log_3 2n \rfloor$  DO
12.   FOR  $\alpha=0:Bound[\beta]$  DO
13.     IF  $\alpha+\beta>0$  THEN
14.       使用引理1计算  $w_n(\alpha,\beta)$  和  $\bar{w}_n(\alpha,\beta)$ 
15.     END IF
16.   END FOR
17.   IF  $n>n_{\alpha,\beta}$  THEN
18.      $w_{\min} \leftarrow \min_V \{2^\alpha 3^\beta + w_n(\alpha,\beta), w_{\min}\}$ 
19.   END IF
20.   IF  $n==n_{\alpha,\beta}$  THEN
21.      $w_{\min} \leftarrow \min_V \{w_n(\alpha,\beta), 2^\alpha 3^\beta + \bar{w}_n(\alpha,\beta), w_{\min}\}$ 
22.   END IF
23. END FOR
24. RETURN  $w_{\min}$ 

```

算法2具有显式二重循环,复杂度为 $O(\log^2 n)$,使用引理1计算 $d_{\alpha,\beta}$ 复杂度为 $O(\log n)$,总复杂度为 $O(\log^3 n)$ 。经实验测得(见5.1.2节),本算法虽然能够得到具有最低理论时值的DBC,但是计算DBC本身的耗时限制了其在零知识证明中的应用,而想要使用DBC得到具有实用价值的椭圆曲线点乘加速算法,需要兼顾DBC质量和计算DBC的耗时。因此本文需要提出一种改进DBC计算时间的方案,在小幅度牺牲DBC质量的前提下大幅提高计算正整数DBC表示的速度。

3.3 次优DBC的求解

算法1可以在确定的时间内计算数字 n 的最优DBC表示,但是其较长的计算时间限制了其应用场景。而DBC计算耗时较长的原因在于,算法需通过动态规划计算所有可能的首项DBC形式,以确保找到最优的DBC表示。而在实际应用中,可以将正整数 n 视为在区间 $[0, p]$ 上的均匀分布,对于大多数随机选取的 n 值,并不需要穷尽所有可能的DBC表示来寻找最优解,而可以通过构建概率模型来优化这一过程。具体来说,可以先分析在哪些条件下,计算出的DBC更有可能得到较低理论时值,然后计算DBC时,只考虑这些条件下的DBC计算,而忽略其他形式的DBC。这种方法虽然会在一定程度上牺牲DBC的质量,但却能显著提高计算DBC的速度,最终实现DBC计算与数乘操作总耗时的最小化。本文将这些理论时值大于最优DBC的DBC称为次优DBC(Sub-Optimal Double Base Chain)。

为了构建次优DBC,本文首先从给定首项条件下的最短DBC出发,Eom等人给出了计算 n 的首项为 $2^\alpha 3^\beta$ 时最短DBC的一个方案^[24],该方案从首项

$2^\alpha 3^\beta$ 开始,逐次找到 α', β' ($\alpha' \leq \alpha, \beta' \leq \beta$)使 $|n - 2^{\alpha'} 3^{\beta'}|$ 尽可能小,然后再计算 $|n - 2^{\alpha'} 3^{\beta'}|$ 的首项为 $2^{\alpha'} 3^{\beta'}$ 的最短DBC,直到 $|n - 2^{\alpha'} 3^{\beta'}| = 0$ 。本文用 $\text{getShortestDBC}(n, \alpha, \beta)$ 函数表示该方案。其中,获得最短DBC的一项时间复杂度为 $O(\log n)$,最短DBC的项数为 $O(\log n)$,因此总复杂度为 $O(\log^2 n)$ 。本文称此算法为Eom-DBC算法,本算法更为详细的说明见^[24]。

二倍点、三倍点和点加的理论时值可以定量刻画它们之间的比例,因此本文可以通过模拟试验分析首项的不同选择策略的DBC的理论时值。在计算理论时值时,本文根据Bernstein等人^[17]给出的结果,取二倍点、三倍点、点加的理论时值分别为 $D=7M, T=12.6M, A=15M$,其中 M 为大整数乘法的时间开销,作为基本单位。为测算对于正整数 n ,如何选定DBC首项能够使该首项的最短DBC更短,本文分别随机生成128 bit、256 bit、384 bit、512 bit的正整数,计算其不同指数首项的DBC表示的理论时值,并对这些结果求平均值。图4给出了DBC的平均理论时值随DBC首项中3的指数的变化关系。

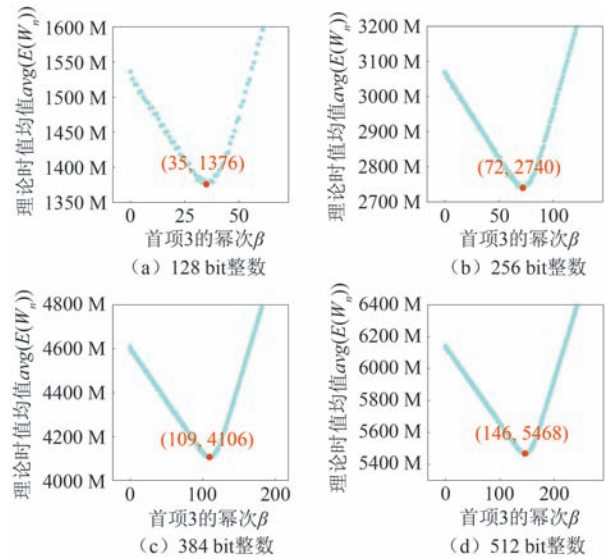


图4 不同首项最短DBC计算平均理论时值

实验显示,DBC表示中首项3的幂次与其理论时值基本呈现一单峰函数关系,例如对于256 bit整数,当首项3的幂次在72左右时,平均理论时值最小。本文使用MinCost数组,依据平均理论时值从小到大对可能的以3为底的幂次进行排序。在计算256位正整数的DBC时,MinCost数组为 $[72, 73, 71, 74, \dots]$ (完整的MinCost数组详见附录)。在得到

MinCost 数组之后,本文依据该数组的顺序来选取 DBC 的首项 3 的幂次进行计算。具体来说,当对一个给定的整数计算 DBC 时,从计算所有可能首项的最短 DBC 修改为只计算 δ 个首项的最短 DBC 时,本文称这种寻找次优 DBC 的方法为 δ -次优 DBC 算法。具体的算法流程如算法 3 所示。

算法 3: 期望理论时值最小的 δ -次优 DBC 算法

输入: 正整数 n

输出: n 的次优 DBC

1. $B_1 \leftarrow \frac{9n}{7\sqrt{2}}, B_2 \leftarrow \frac{16\sqrt{2}n}{21}, w_{\min} \leftarrow \text{NAF}(n)$
2. $count \leftarrow 0$
3. FOR EACH β in $MinCost$ DO
4. $LBound[\beta] \leftarrow \left\lceil \log \frac{B_1}{3^\beta} \right\rceil, RBound[\beta] \leftarrow \left\lceil \log \frac{B_2}{3^\beta} \right\rceil$
5. IF $LBound[\beta] == RBound[\beta]$ THEN
6. $c \leftarrow \text{getShortestDBC}(n, RBound[\beta], \beta)$
7. $w_{\min} \leftarrow \min(c, w_{\min})$
8. $count \leftarrow count + 1$
9. IF $count == \delta$ THEN
10. break
11. END IF
12. END IF
13. END FOR
14. RETURN w_{\min}

在算法 3 中, δ -次优 DBC 算法给出了 δ 个可能的 DBC 首项,总复杂度为 $O(\delta \log^2 n)$ 。同时,考虑最坏情况,由于 w_{\min} 初始化为 n 的 NAF 表示,所以在计算椭圆曲线点乘部分,使用 δ -次优 DBC 算法计算得到的 DBC 完成椭圆曲线数乘的计算速度不会慢于使用 NAF 计算的速度。

4 GPU 并行

GPU 已经成为当今主流计算系统的一个组成部分。现代 GPU 不仅是一个强大的图形引擎,而且是一个高度并行的可编程处理器,一直被研究者广泛应用于高性能、高并发计算的场景中。现在越来越多的研究者开始运用 GPU 加速密码学算法的计算。

传统密码学方案多用于通信协议的底层安全支持,较少成为性能瓶颈,因此引入异构计算来加速其意义不大。然而, zk-SNARK 的性能瓶颈却非常适合通过 GPU 并行计算的方式来加速。根据公式(3)给出的椭圆曲线多项式的一般计算形式,由于 x 在

zk-SNARK 协议中由可信第三方生成,并且对验证者和证明者保密,因此证明者需要代入 $\{x^i * g\}$ 的具体值来计算 $[\Pi](x)$ 的值。 $\{c_i * (x^i * g)\}$ 的每一项之间不存在数据依赖关系,因此可以采用 SIMD 模式在 GPU 上进行并行计算。

zk-SNARK 验证证明的步骤仅需一次确定复杂度的椭圆曲线配对计算,而生成证明的时间依照电路规模确定,更加耗时,因此实际的协议性能受到生成证明的时间的限制。图 1 给出了测试后 Groth16 各个步骤的操作数和计算时间比例。在生成证明阶段,主要需要进行多项式乘法计算和多项式承诺的计算。计算多项式承诺的过程在椭圆曲线域上进行,该过程约占总生成证明时间的 65%。作为对比,验证 zk-SNARK 证明的时间不受问题规模影响,从而耗时较小。可以看出, zk-SNARK 的性能瓶颈就在于计算多项式承诺阶段,也就是公式(1)给出的计算 $[\chi]_1, [\psi]_2, [\omega]_1$ 的内容。

在 Groth16 协议当中,生成证明步骤需要计算 χ, ψ, ω 的值,公式(1)给出了该计算的过程,此计算涉及椭圆曲线高次多项式的计算,此过程需要计算几万个椭圆曲线点乘,因此本文可以使用 GPU 并行加速的方法来并行计算一个高次椭圆曲线多项式。图 5 展示了 GPU 计算多项式承诺的架构图。

4.1 GPU 密码学计算库

为实现 GPU 上的椭圆曲线多项式并行计算,首先需要实现一个底层支持椭圆曲线基础操作的计算库。Robert Szwed 等人首次给出了利用 GPU 计算平台构建密码学计算库^[32]的方案,但该方案仅能在模大素数有限域上进行密码学计算,如完成 RSA 公钥加密。2024 年,高钰洋等^[33]设计了基于 GPU 的椭圆曲线运算库,本文在此运算库基础上集成了 DBC 技术来加速椭圆曲线上的数乘运算,给出了使用 SIMD 模式对 zk-SNARK 的 MSM 计算过程进行优化的并行计算方案。

考虑 GPU 计算架构中并不原生支持多字节整数,因此实现大整数计算库时,最需要关注如何快速实现除法和求逆两种运算。已有研究指出^[17],对于多字节整数, GPU 上直接计算大整数模数除法的消耗约为计算大整数乘法的 100 倍。在计算 DBC 时,所有分支条件和余项计算需要进行大量的除法,本文必须降低这一开销。为此,本文引入了 Jacobi 射影坐标系,并在射影坐标系上将元素映射到蒙哥马利域上计算。蒙哥马利域需要选定一个与有限域模

数 p 互素的模数 p_{any} , 对于元素 a 而言, 其蒙哥马利域上对应的元素即为 $Mont(a) = a \cdot p_{any}$. 在蒙哥马利域上, 乘法定义如公式(4)所示:

$$Mont_mul(Mont(a), Mont(b)) = Mont(a) \cdot Mont(b) \cdot p_{any}^{-1} \quad (4)$$

易证 $Mont(a, b) = Mont(a) \cdot Mont(b)$, 这表明了蒙哥马利域对乘法具有封闭性. 使用蒙哥马利域, 可以将一个大整数对随机模数 p 的计算转换成对于选定的任意模数 p_{any} 的取模. 本文可以令 p_{any} 取到一些特殊的值, 例如在 256 位椭圆曲线上令 $p_{any} = 2^{256}$, 这样, 模 p_{any} 的计算就变为简单的位运算, 通过这种方式可以大幅加快模 p 除法计算速度, 即使用射影坐标系下的蒙哥马利域, 本文仅需要三到四次乘法的时间

即可进行求逆计算. 鉴于本文 GPU 上的 zk-SNARK 均在 BN128 椭圆曲线上运行, 该椭圆曲线模数为 254 位, 因此其处理的大整数不会超过 256 位.

Savas 等人提出了蒙哥马利求逆算法的一个实现^[34], 本文利用此算法实现了大整数的求逆. GPU 密码学计算库的具体构建方式见文献[33].

4.2 GPU 计算架构

在 zk-SNARK 生成证明的计算过程中, 电路生成、规约 QAP 问题和聚合多项式的过程具有较强的数据依赖性, 难以利用 GPU 的并行计算能力进行大规模 GPU 端计算. 因此本文 GPU 端计算仅应用于计算多项式承诺步骤的大规模椭圆曲线数乘步骤. 本文通过两个阶段的计算来实现并行求大规模椭圆曲线多项式结果的能力(如图 5 所示):

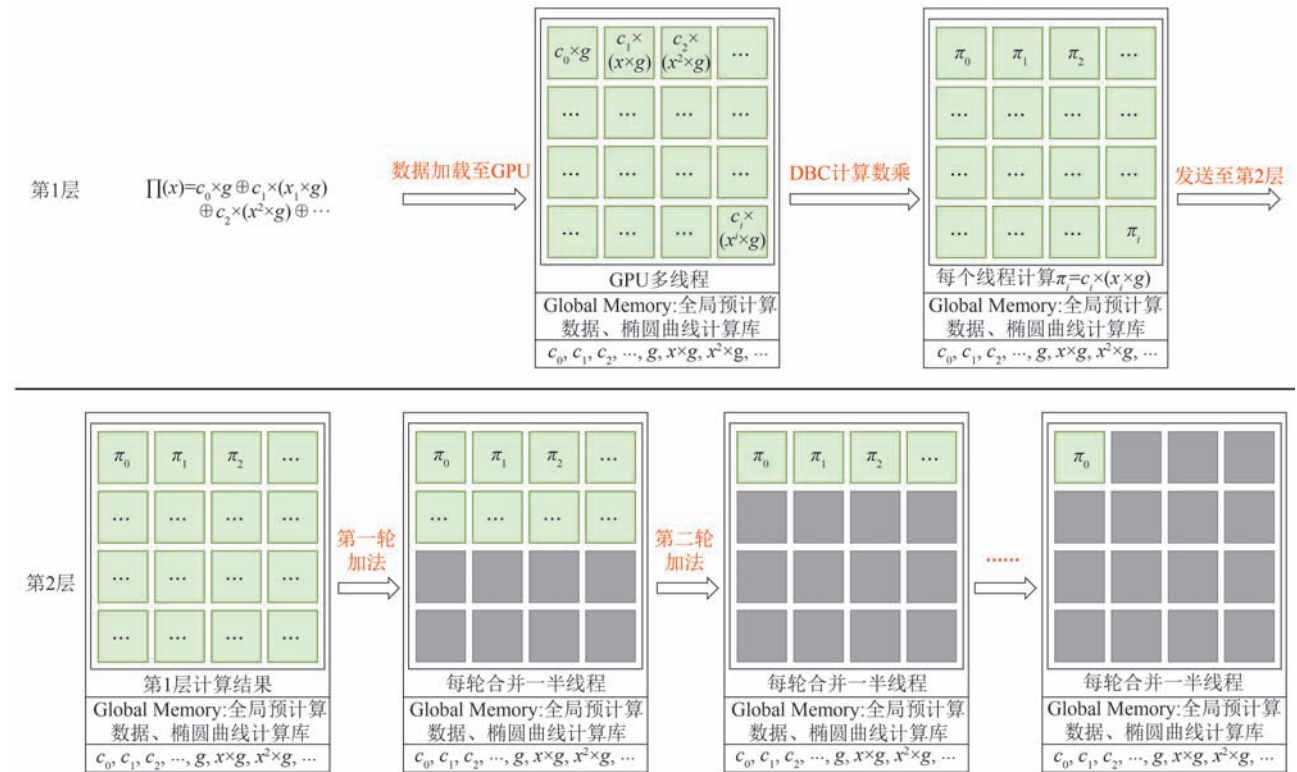


图5 GPU 计算多项式承诺架构

第一阶段: 通过将多项式的每一项分配到 GPU 的每一个线程上, 在该线程上对椭圆曲线数乘的乘数运行 DBC 求解算法, 得到乘数的 DBC 后根据 DBC 执行椭圆曲线乘法, 该过程会运行若干次椭圆曲线二倍点操作、椭圆曲线三倍点操作和椭圆曲线加法操作.

第二阶段: 根据第一阶段的结果, 本文得到等同于多项式非零项数数量的椭圆曲线点(为第一阶段数乘计算的结果), 本文通过每轮合并两个线程的结果, 经由 $\log(m)$ 轮加法计算得到 m 个椭圆曲线点的

加和结果, 得到椭圆曲线多项式的值. 具体地, 在第 k 轮本文将计算:

$$\pi_i = \pi_i \oplus \pi_{i+m/2^k}, 0 \leq i \leq \frac{m}{2^k}.$$

其中, π_i 是第一阶段计算的 $c_i \times (x^i \times g)$ 的结果.

本文进一步说明 GPU 上进行快速加法的流程. 以 16 个点的加法为例, 如图 6 所示: 直接计算 16 个点的和需要 15 次点加操作, 但在并行计算框架下, 本文每一轮可以将点两两分配, 从而每一轮能够将

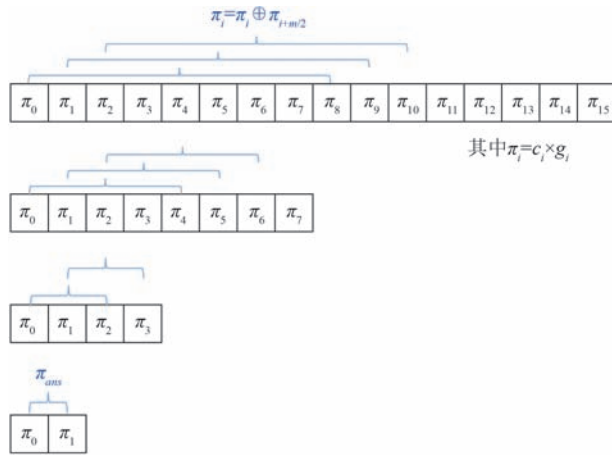


图6 GPU计算多项式承诺架构

待计算的点的数量减少一半,最终本文仅需4轮加法就可以计算出最终结果。用这种并行计算计算 m 个点的加法,本文只需要 $\log(m)$ 轮加法即可以计算出最后的结果,实现了计算加速。

在实际操作中,对于在GPU上运行的计算任务,首先将数据由CPU端的RAM传输到GPU端内存。完成计算后,结果会再次被传回CPU端以供下一步计算使用。同时,为了减少重复计算,在计算DBC的结果时,本文会预先计算好一系列 $2^a 3^b$ 的值,并把这些值存储于公共内存区(GPU端的shared memory)。这样在后续的DBC计算过程中可以直接使用这些预存的值完成计算。

表4 使用不同算法所求次优DBC表示计算数乘步骤耗时数据(M为有限域上数字乘法,下同)

DBC求解算法	理论时值 $V(w_n)$	数乘步骤平均耗时(μs)	单步骤理论加速比	单步骤实验加速比
最优DBC算法	2586 M	141.8	118.10%	118.00%
Eom-DBC算法 ^[25]	2677 M	144.4	114.10%	115.90%
1-次优DBC算法	2704 M	146.6	112.90%	114.10%
2-次优DBC算法	2694 M	146.2	113.40%	114.40%
3-次优DBC算法	2687 M	146.0	113.70%	114.60%
10-次优DBC算法	2677 M	145.6	114.10%	114.90%
20-次优DBC算法	2676 M	145.5	114.10%	115.00%
二进制计算算法	3704 M	189.6	82.50%	88.20%
NAF算法(baseline)	3054 M	167.3	100.00%	100.00%

从表4中理论加速比和实验加速比的比较可以看出,通过理论时值来刻画实际的耗时基本可以反映真实实验数据的比较情况。从选择不同DBC算法的具体耗时上可以看出,使用最优DBC算法计算出的DBC有着最小的理论时值和最快的数乘计算速度,Eom-DBC算法次之,而对于 δ -次优DBC算法, δ 的取值越小,计算出的DBC理论时值越高,数

5 实验测试与结果

本章通过实现分析对本方案进行性能测试,实验运行于Ubuntu 20.04系统上,CPU型号为Intel i9-12900KF,GPU型号为Nvidia 4090,CUDA版本为12.4。

5.1 DBC算法性能测试

本节通过实验数据讨论本文提出的算法对于DBC的计算效率和DBC质量的影响,以说明本文的工作可以实现二者的取舍,通过略微降低DBC质量大幅提高计算正整数DBC表示的效率,从而使得整体的计算效率得到提升。本小节所使用的椭圆曲线为secp256k1曲线。

5.1.1 不同DBC形式的数乘步骤计算耗时分析

本节中使用理论时值来衡量了DBC的质量,同时DBC求解算法也通过理论时值作为动态规划DBC表示的方法。理论时值刻画了一个表示(DBC形式、NAF形式或二进制形式)的理论数乘步骤计算时间,表4给出了不同的DBC求解算法表示的理论时值和实际测得的数乘步骤计算时间结果。实验数据来源为随机生成10 000个256 bit整数,使用不同DBC求解算法和NAF算法、二进制形式分别求其不同表示的理论时值,并以此形式计算数乘以求得实际计算耗时,取平均得到数据。

乘计算时间也越长。但从总体数据上来看,选择较小的 δ 取值的次优DBC计算倍点的耗时增加的时间并不长(最差为 $146.6 - 141.8 = 4.8 \mu\text{s}$),这相对于整体的计算时间来说并不会引起显著的性能退化。

5.1.2 使用DBC算法整体计算时间分析

对于原始的DBC算法来说,计算DBC形式的时间可能比使用DBC形式计算数乘的时间更长。

但在使用DBC计算倍点的过程中,通常也需要考虑计算大整数DBC的时间。因此本节将计算DBC表示的时间也纳入实验中,通过随机生成10 000个256位的正整数,并用不同的方法计算其DBC表示,记录其计算的平均时间。其中最优DBC采用算法2

进行计算,次优DBC算法均采用算法3进行计算。本文统计不同的DBC计算算法占整体计算数乘操作的时间比例和与目前常用的NAF算法(OpenSSL等通用计算库所使用的数乘算法)、二进制算法相比的加速比,实验结果如表5所示。

表5 不同DBC算法完整计算时间测试

DBC求解算法	理论时值 $V(w_n)$	数乘步骤平均耗时(μs)	单步骤理论加速比	单步骤实验加速比
最优DBC算法	2586 M	141.8	118.1%	118.0%
Eom-DBC算法 ^[24]	2677 M	144.4	114.1%	115.9%
1-次优DBC算法	2704 M	146.6	112.9%	114.1%
2-次优DBC算法	2694 M	146.2	113.4%	114.4%
3-次优DBC算法	2687 M	146.0	113.7%	114.6%
10-次优DBC算法	2677 M	145.6	114.1%	114.9%
20-次优DBC算法	2676 M	145.5	114.1%	115.0%
二进制计算算法	3704 M	189.6	82.5%	88.2%
NAF算法(baseline)	3054 M	167.3	100.0%	100.0%

从表5中可以看出,对于最优DBC和一部分 δ -1次优DBC而言,使用DBC算法来计算数乘的总时间开销实际上大于NAF算法,尽管前述实验说明了对于单独的数乘步骤而言最优DBC有约18%的性能优势。其原因在于,对于传统DBC算法而言计算DBC本身的开销较大。尽管传统DBC算法的理论复杂度是确定的多项式级别(相对于数字的比特数而言),但仍然难以满足实际的应用场景。作为对比,1-次优DBC算法计算DBC表示的速度相对最优DBC可以做到约33倍的加速,这令计算DBC形式这一原本耗时巨大的步骤现在只占总体时间的5.2%,而从包含计算DBC形式和使用DBC形式计

算数乘两个步骤整体上来看,使用1-次优DBC算法相比于NAF算法计算速度约快8.2%。这意味着本文对于DBC算法的优化工作可以将DBC应用于通用的椭圆曲线计算库中。

5.1.3 不同规模多项式数乘计算性能测试

本节对基于CPU的椭圆曲线数乘进行实验,所采用的椭圆曲线库为OpenSSL,所使用的椭圆曲线是secp256k1。实验过程中,本节将椭圆曲线多项式的次数分别设置为从 2^{10} 到 2^{17} ,以测试使用不同算法计算椭圆曲线点乘所需的时间。

如表6所示,在 2^{10} 至 2^{17} 次椭圆曲线多项式的计算上,1-次优DBC算法均能取得最佳性能。

表6 不同计算方法计算椭圆曲线点乘的用时

多项式次数	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
最优DBC算法(ms)	429	863	1722	3418	6796	13 541	27 045	54 057
Eom-DBC算法(ms)	301	602	1205	2427	4851	9699	193 76	38 742
1-次优DBC算法(ms)	159	318	639	1275	2548	5101	10 201	20 430
二进制算法(ms)	196	392	782	1571	3141	6297	12 565	25 107
NAF算法(ms)	177	355	712	1423	2849	5703	11 407	22 914

5.1.4 不同椭圆曲线的DBC计算性能测试

同时,为了证明次优DBC算法的有效性,本文在四条不同的椭圆曲线进行实验。随机生成10 000个的128 bit、256 bit、384 bit、521 bit整数并采用不同算法计算椭圆曲线点乘,测试各算法计算点乘的用时。测试结果如表7所示:

如表7所示,对于表中的四条椭圆曲线,使用1-次优DBC算法均能够得到最快的计算速度。

表7 不同计算方法计算椭圆曲线点乘的用时

椭圆曲线	secp128r1	secp256k1	secp384r1	secp521r1
最优DBC算法(ms)	1217	4057	8948	17 072
Eom-DBC算法 ^[24] (ms)	830	2887	6728	16 334
1-次优DBC算法(ms)	559	1546	2971	6362
NAF算法(ms, baseline)	620	1673	3281	6936
二进制算法(ms)	648	1896	3773	8341

5.2 GPU性能测试

5.2.1 与CPU方案的对比

本章节的DBC算法采用1-次优DBC算法(见算法3),同时本节使用CPU上运行的libsark库作为对比实验。libsark提供了完整的Groth16协议实现,在多项式承诺阶段,libsark使用Pippenger算法计算椭圆曲线多项式。Pippenger算法是Nicholas Pippenger提出的在通用计算平台上实现的用于同时计算多个椭圆曲线数乘操作的算法,目前仍然是主流的零知识证明实现方案中对于多项式承诺计算的通用实现方案。本节实验所采用的椭圆曲线为BN128曲线。

GPU端测试结果如图7所示。针对于点的数量较少的情況(少于 2^{12} 即4096个点),两者的差距并不明显。而对于现实场景中有意义的零知识证明问题而言,通常其多项式承诺项数会达到数万项(多于 2^{15} 个点),对于这种规模的计算,GPU平台便可以充分利用其并行能力,表现出明显的优势。在椭圆曲线多项式次数为 2^{20} 时,多项式承诺阶段计算速度可达libsark的5.76倍。按照椭圆曲线点乘时间占比65%计算,多项式承诺的5.76倍加速相当于生成证明全过程的2.16倍加速。

5.2.2 与Lu等人^[30]方案的对比

Lu等人^[30]设计了基于Pippenger算法的处理大规模椭圆曲线点乘的方案,该方案在计算高次椭圆

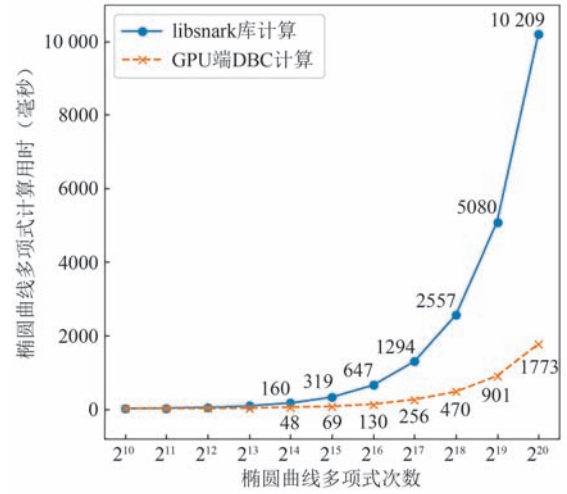


图7 GPU测试结果(对比 libsark)

曲线多项式时有着较快的计算速度,但该算法无法良好处理稀疏椭圆曲线多项式。记公式3所示的椭圆曲线多项式次数为 d ,其汉明重量(c_0, c_1, \dots, c_d 中非零元素个数)为 ξ 。本节从稠密椭圆曲线多项式和稀疏椭圆曲线多项式两个角度出发将本文的方案与Lu等人的方案^[30]进行对比。

(1)稠密椭圆曲线多项式,即 $\xi = d$,实验结果如表8所示:

如表8所示,本方案在稠密椭圆曲线多项式上计算速度慢于Lu等人的方案,这表明本方案具有一定的局限性。

表8 稠密椭圆曲线多项式计算时间对比

MSM多项式次数	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
本文的方案用时(ms)	25	48	69	130	256	470	901	1773
Lu等人 ^[30] 的方案用时(ms)	22	23	25	25	26	32	34	48

(2)稀疏椭圆曲线多项式,本文取 $\xi = \lceil \sqrt{d} \rceil$, $\xi = \lceil d/100 \rceil$, $\xi = \lceil d/1000 \rceil$ 作为稀疏多项式的标准,实验结果如表9所示:

根据表9所示的数据,Lu等人的方案在计算次数为 2^{13} 至 2^{17} 次方的稀疏椭圆曲线多项式时速度大幅慢于稠密椭圆曲线多项式,这说明此时椭圆曲线多项式计算为该方案的较坏情况,而对于次数为 2^{18} 至 2^{20} 的稀疏椭圆曲线多项式,Lu等人的方案可以正常完成计算工作,但其计算速度与稠密椭圆曲线多项式基本持平。而本方案在完成稀疏椭圆曲线多项式的椭圆曲线点乘工作时,计算速度相比于Lu等人的方案有优势。

因此,本方案适用于需要高效处理大量稀

疏椭圆曲线多项式的实际应用场景,例如BulletProofs++(BP++)^[35]范围证明协议。作为目前已知计算性能最优、证明大小最小的范围证明协议,BP++的核心思想之一是通过多轮折叠私有见证向量来缩减证明大小(每轮折叠使向量长度减半)。在每一轮计算中,需要执行大量的椭圆曲线数乘运算,这些运算可转化为多个稀疏椭圆曲线多项式的计算。具体的,比如计算典型运算 $ag + bh$,其中 $a, b \in \mathbb{Z}_p^*$, $g, h \in \mathbb{G}$,该椭圆曲线加法群的阶数为素数 p 。从初始化信任设置CRS集合 $x^i * g$ 的角度,我们视 $h = x^0 * g$,因 x 未知, θ 极大概率是一个高次数。于是 $ag + bh$ 可视为一个稀疏多项式。

值得注意的是,不同轮次产生的稀疏椭圆曲线

表9 稀疏椭圆曲线多项式计算时间对比

(a) $\xi = \lceil \sqrt{d} \rceil$								
MSM多项式次数	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
本文的方案用时(ms)	14	14	15	15	16	18	20	28
Lu等人 ^[30] 的方案用时(ms)	66	70	80	95	133	32	36	51
(b) $\xi = \lceil d/100 \rceil$								
MSM多项式次数	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
本文的方案用时(ms)	13	14	15	16	29	45	78	150
Lu等人 ^[30] 的方案用时(ms)	67	71	78	94	144	33	35	49
(c) $\xi = \lceil d/1000 \rceil$								
MSM多项式次数	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
本文的方案用时(ms)	13	13	14	14	15	17	20	28
Lu等人 ^[30] 的方案用时(ms)	66	69	78	98	133	33	35	49

多项式无法聚合。这是由于每一轮验证者都会生成一个新的挑战值发送给证明者,若强行聚合这些跨轮次的多项式,将破坏整个证明系统的健全性。

最后一点,设稀疏多项式 $\Pi_1(x), \Pi_2(x)$ 的最高次数为 d , 非零项数分别为 $\xi_1 = \rho_1 d, \xi_2 = \rho_2 d$ 。其中稀疏度 ρ_1, ρ_2 满足 $0 < \rho_1 \leq \rho_2 \leq 1$ 。那么 $\Pi_1(x) + \Pi_2(x)$ 的稀疏度为 $O\left(\frac{\rho_1 d + \rho_2 d}{d}\right) = O(\rho_1 + \rho_2)$, $\Pi_1(x) \cdot \Pi_2(x)$ 的稀疏度为 $O\left(\frac{\rho_1 d \cdot \rho_2 d}{2d}\right) = O\left(\frac{\rho_1 d \cdot \rho_2}{2}\right)$ 。直观说起来,稀疏多项式的加法为稀疏;稀疏多项式的乘法为稀疏,若其中稀疏多项式 $\Pi_1(x)$ 非零项数 $\rho_1 d$ 为一个小规模常数。这种复合规则会指导我们处理稀疏椭圆曲线多项式的计算。

6 结 论

本文研究了 zk-SNARK 中多项式承诺步骤的计算加速方案。现有的 zk-SNARK 实现方案在生成证明的过程中,最耗时的步骤就是计算多项式承诺,因为该步骤需要进行大量的椭圆曲线域的计算,在实际应用中该步骤的操作数通常能达到数十万项或更多。由于计算多项式承诺的步骤就是计算一个大规模椭圆曲线多项式的结果,本文从两个层面对多项式承诺的计算过程进行了加速。

第一,本文使用 DBC 形式来表示单步椭圆曲线数乘计算当中的乘数,相比于传统的二进制或者 NAF 表示,将乘数以 DBC 形式计算的开销会更低。而针对现有 DBC 研究中计算 DBC 形式耗时

过大的问题,本文提出了一系列次优 DBC 算法来达到 DBC 计算效率和 DBC 质量的平衡。在 Intel i9-12900KF CPU 上,次优 DBC 算法计算 DBC 形式仅占据完整数乘计算时间的 5.2%,相比于原始 DBC 算法计算速度提升了 33 倍,而整体计算速度相比于 OpenSSL 的实现提升了 8.2%。同时,在最坏情况下,由于计算次优 DBC 时,首先将数乘倍数 n 初始化为 NAF 表示,所以次优 DBC 算法不会慢于 NAF 算法的计算速度,这保证了次优 DBC 算法在性能上的稳定性。

第二,针对大规模椭圆曲线多项式的计算,本文利用了 GPU 平台计算来并行计算椭圆曲线多项式,通过利用 GPU 平台相比于通用计算平台更强的并行能力,针对数十万项的椭圆曲线多项式计算任务,本文基于 Nvidia CUDA 实现了椭圆曲线的基础计算库,并在其上实现了 DBC 算法。最好情况下,在 Nvidia 4090 GPU 上,相比于 libsnark 的多项式承诺实现,本文的 GPU 实现时间实现了 5.76 倍的加速。与 Lu 等人^[30]给出的基于 Pippenger 算法的椭圆曲线点乘方法对比,本文的方案在椭圆曲线多项式较为稀疏时有着明显的优势。而未来本文将考虑结合本方案与 Lu 等人的方案的优势。

本文基于计算多项式承诺的应用提出了针对 DBC 计算方案的优化,并将其在 GPU 上实现。在本文的工作之上,仍然有许多值得深入挖掘的方向。从实验结果可以看出,相比于已有的方案,本文的 GPU 方案在椭圆曲线多项式比较稀疏的情形有着明显的优势,但当椭圆曲线多项式较为稠密时,本文的方案相比于已有方案仍有不足之处。未来本文

将考虑结合稠密椭圆曲线多项式和稀疏椭圆曲线多项式的计算优势。此外,本文只在GPU上实现了bn128一条椭圆曲线,未来也将考虑扩展到其他椭圆曲线。而在次优DBC计算方面,本文未来会考虑研究MinCost数组的分布。读者可访问https://github.com/scalarMSM/DBC_snark仓库以获得本文的测试代码和测试样例。

致谢 感谢公共大数据国家重点实验室开放课题(PBD2022-12)、天津市科技计划重点研发计划(19YFZCSF00900, 20JCZDJC00610)、国家自然科学基金项目(62272253)对本课题的资助。感谢许友锐在论文修改过程中的协助工作。

参 考 文 献

- [1] Goldwasser S, Micali S, Rackoff C. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 1989, 18(1): 186-208
- [2] Bünz B, Bootle J, Boneh D, et al. Bulletproofs: Short proofs for confidential transactions and more//*Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP)*. San Francisco, USA, 2018: 315-334
- [3] Zhang J, Su M, Liu X, et al. Springproofs: Efficient inner product arguments for vectors of arbitrary length//*IEEE Symposium on Security and Privacy*. San Francisco, USA: 2024: 3147-3164
- [4] Hopwood D, Bowe S, Hornby T, et al. Zcash Protocol Specification. San Francisco, USA: GitHub, 2016: 1-220
- [5] ProjectMonero. Monero research lab [repository]. <https://github.com/monero-project/research-lab/tree/master>.
- [6] Gluchowski A. Zk rollup: Scaling with zero-knowledge proofs. Matter Labs, 2019.
- [7] Garoffolo A, Kaidalov D, Oliynykov R. Zendo: A zk-SNARK verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains//*Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. New York, USA, 2020: 1257-1262
- [8] Zhang J, Fang Z, Zhang Y, et al. Zero knowledge proofs for decision tree predictions and accuracy//*Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. New York, USA, 2020: 2039-2053
- [9] Pan Y, Zhao Y, Liu X, et al. FPLotto: A fair blockchain-based lottery scheme for privacy protection//*Proceedings of the 2022 IEEE International Conference on Blockchain (blockchain)*. New York, USA, 2022: 21-28
- [10] Zhang Y, Genkin D, Katz J, et al. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases//*Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*. San Francisco, USA, 2017: 863-880
- [11] Zhao Z, Chan T H H. How to vote privately using bitcoin//*International Conference on Information and Communications Security*. Cham, Switzerland: Springer, 2015: 82-96
- [12] Groth J. On the size of pairing-based non-interactive arguments//*Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Berlin, Germany, 2016: 305-326
- [13] NIST U. S. Department of Commerce. Descriptions of SHA-256, SHA-384 and SHA-512. Gaithersburg, USA: National Institute of Standards; Technology (NIST), 2001
- [14] Merkle R C. Method of providing digital signatures. US Patent 4,309,569, 1982-12-15
- [15] FileCoin. <https://filecoin.io>
- [16] Gennaro R, Gentry C, Parno B, et al. Quadratic span programs and succinct NIZKs without PCPs//*Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Berlin, Germany, 2013: 626-645
- [17] Bernstein D J. Explicit-formulas database. <http://www.hyperelliptic.org/EFD>
- [18] Blake I F, Murty V K, Xu G. A note on window τ -NAF algorithm. *Information Processing Letters*, 2005, 95(5): 496-502
- [19] Doche C, Kohel D R, Sica F. Double-base number system for multi-scalar multiplications//*Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Berlin, Germany, 2009: 502-517
- [20] Adikari J, Dimitrov V S, Imbert L. Hybrid binary-ternary number system for elliptic curve cryptosystems. *IEEE Transactions on Computers*, 2010, 60(2): 254-265
- [21] Doche C, Sutantyo D. New and improved methods to analyze and compute double-scalar multiplications. *IEEE Transactions on Computers*, 2012, 63(1): 230-242
- [22] Yu W, Musa S A, Li B. Double-base chains for scalar multiplications on elliptic curves//*Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Cham, Switzerland, 2020: 538-565
- [23] Dimitrov V S, Imbert L, Mishra P K. Efficient and secure elliptic curve point multiplication using double-base chains//*International Conference on the Theory and Application of Cryptology and Information Security*. Berlin, Germany, 2005: 59-78
- [24] Eom S-H, Lee H-S, Lim S, et al. Analysis on yuet al.'s dynamic algorithm for canonic DBC. *Discrete Applied Mathematics*, 2021, 294: 31-40
- [25] Zhang Y, Wang S, Zhang X, et al. Pipezk: Accelerating zero-knowledge proof with a pipelined architecture//*Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. New York, USA, 2021: 416-428
- [26] Xie M, Hao M, Yang H, et al. Survey on hardware acceleration for zero-knowledge proofs. *Information Security Research*, 2024, 10(07): 594-601
- [27] Ding D, Li Z Q. Research on hardware acceleration methods for elliptic curve operations in zero-knowledge proofs. *Journal of China Jiliang University*, 2024, 35(02): 185-196 (in Chinese) (丁冬, 李正权. 零知识证明中椭圆曲线运算的硬件加速方法研究. *中国计量大学学报*, 2024, 35(02): 185-196)

- [28] Manavski S A. CUDA compatible GPU as an efficient hardware accelerator for AES cryptography//Proceedings of the 2007 IEEE International Conference on Signal Processing and Communications. Dubai, UAE, 2007: 65-68
- [29] Ni N, Zhu Y. Enabling zero knowledge proof by accelerating zk-SNARK kernels on GPU. Journal of Parallel and Distributed Computing, 2023, 173: 20-31
- [30] Lu T, Wei C, Yu R, et al. CUZK: Accelerating zero-knowledge proof with a faster parallel multi-scalar multiplication algorithm on GPUs. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023, 2023(3): 194-220
- [31] Ma W, Xiong Q, Shi X, et al. GZKP: A GPU accelerated zero-knowledge proof system//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2023, 2: 340-353
- [32] Szerwinski R, Güneysu T. Exploiting the power of GPUs for asymmetric cryptography//International Workshop on Cryptographic Hardware and Embedded Systems. Berlin, Germany, 2008: 79-99
- [33] Gao Yuyang, Zhang Jianning, WANG Gang, et al. Optimization for GPU-based elliptic curve library and related algorithms. Journal of Cyber Security, 2024, 9(06): 1-16(in Chinese)
(高钰洋, 张健宁, 王刚等. 基于GPU的椭圆曲线运算库及相关算法优化. 信息安全学报, 2024, 9(06): 1-16)
- [34] Savas E, Kaya Koç Çetin. The montgomery modular inverse-revisited. IEEE Transactions on Computers, 2000, 49(7): 763-766
- [35] Eagen L, Kanjalkar S, Ruffing T, et al. Bulletproofs++: Next generation confidential transactions via reciprocal set membership arguments//Annual International Conference on the Theory and Applications of Cryptographic Techniques. 2024: 249-279

附录 A. MinCost 数组具体取值

计算 128 bit 椭圆曲线点乘时, MinCost 数组为

[35, 36, 34, 37, 33, 38, 32, 39, 31, 30, 40, 29, 41, 28, 27, 42, 26, 43, 25, 24, 44, 23, 22, 45, 21, 20, 46, 19, 18, 47, 17, 16, 48, 15, 14, 49, 13, 12, 50, 11, 10, 51, 9, 8, 7, 52, 6, 5, 53, 4, 3, 54, 2, 1, 55, 0, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81].

计算 256 bit 椭圆曲线点乘时, MinCost 数组前 100 项为

[72, 73, 71, 74, 70, 75, 69, 76, 68, 67, 77, 66, 78, 65, 79, 64, 80, 63, 62, 81, 61, 82, 60, 59, 83, 58, 57, 84, 56, 55, 85, 54, 86, 53, 52, 87, 51, 50, 88, 49, 48, 47, 89, 46, 45, 90, 44, 43, 91, 42, 41, 92, 40, 39, 93, 38, 37, 36, 94, 35, 34, 95, 33, 32, 96, 31, 30, 29, 97, 28, 27, 98, 26, 25, 99, 24, 23, 100, 22, 21, 20, 101, 19, 18, 102, 17, 16, 103, 15, 14, 104, 13, 12, 11, 105, 10, 9, 106, 8, 7].

计算 384 bit 椭圆曲线点乘时, MinCost 数组前 100 项为

[109, 110, 108, 111, 107, 112, 106, 105, 113, 104, 114, 103, 115, 102, 116, 101, 117, 100, 99, 118, 98, 119, 97, 120, 96, 95, 121, 94, 122, 93, 92, 123, 91, 124, 90, 89, 125, 88, 87, 126, 86, 85, 127, 84, 83, 128, 82, 81, 129, 80, 79, 130, 78, 77, 131, 76, 75, 132, 74, 73, 72, 133, 71, 70, 134, 69, 68, 135, 67, 66, 136, 65, 64, 63, 137, 62, 61, 138, 60, 59, 139, 58, 57, 56, 140, 55, 54, 141, 53, 52, 142, 51, 50, 143, 49, 48, 47, 144, 46, 45].

计算 512 bit 椭圆曲线点乘时, MinCost 数组前 100 项为

[146, 147, 145, 148, 144, 149, 143, 150, 142, 151, 141, 152, 140, 153, 139, 138, 154, 137, 155, 136, 156, 135, 157, 134, 133, 158, 132, 159, 131, 130, 160, 129, 161, 128, 127, 162, 126, 163, 125, 124, 164, 123, 122, 165, 121, 166, 120, 119, 167, 118, 117, 168, 116, 115, 169, 114, 113, 170, 112, 111, 110, 171, 109, 108, 172, 107, 106, 173, 105, 104, 174, 103, 102, 101, 175, 100, 99, 176, 98, 97, 177, 96, 95, 94, 178, 93, 92, 179, 91, 90, 180, 89, 88, 87, 181, 86, 85, 182, 84, 83].



WEN Zhou-Zhi, M. S. His research mainly focuses on cryptography, blockchain technology, and zero-knowledge proofs.

JIANG Ze-En, M. S. His research mainly focuses on blockchain covert channels.

YANG Hao-Ran, M. S. His research mainly focuses on zero-knowledge proofs and fully homomorphic encryption.

SU Ming, Ph. D., associate researcher. His research interests include sequence complexity and related algorithms, digital watermarking, and blockchain.

LIU Xiao-Guang, Ph. D., professor. His research interests include search engines, storage systems, and GPU computing.

Background

Since Goldwasser et al. introduced zero-knowledge proofs in 1985, zero-knowledge proofs have attracted widespread attention, and their related work got the Turing Award in 2012.

Zero-knowledge proof schemes are powerful cryptographic tools that can be widely applied in various cryptographic protocols involving security and privacy information. However, the long time required to generate zero-knowledge proofs limits their practical applications. The root cause of this issue is that generating zero-knowledge proofs requires the computation of a large-scale elliptic curve polynomial, which accounts for 65% of the total proof generation time. Mainstream open-source projects often use the Pippenger algorithm to compute this step. In contrast, this paper proposes a DBC solution to compute single-step elliptic curve operations combined with GPU parallel computing for large-scale elliptic curve polynomial computation.

The proposed method outperforms widely used open-source cryptographic libraries in terms of theoretical computation time and test results.

Our implementation is approximately 8.2% faster than OpenSSL's for single-step elliptic curve scalar multiplication. For the computation of polynomial commitment, our GPU implementation achieves a 5.76x speedup for large-scale elliptic curve polynomials (over 2^{15} points) compared to on an Nvidia 4090 GPU. (libsark implements zk-SNARKs such as Groth16, which has been incorporated into national blockchain cryptography standards).

This paper further improves upon existing DBC computation schemes by introducing a suboptimal DBC algorithm. This algorithm accelerates the time required to calculate expensive DBC solution steps by 33 times without significantly affecting the quality of DBC.