基于 GPU 的 X25519/448 密钥协商算法的高速实现

董建阔[1,2,3], 郑昉昱[1,2], 林璟锵[1,2,3]

[1] 中国科学院信息工程研究所 信息安全国家重点实验室 北京 中国 100093
[2] 中国科学院数据与通信保护研究教育中心 北京 中国 100093
[3] 中国科学院大学网络空间安全学院 北京 中国 100049

摘要 密钥协商算法目前被广泛运用于包括TLS/SSL在内的各种安全协议中, 以支持通信双方在不被保护的信道中建立共享秘密。特别是在 TLS 1.3 中, 为保证前向安全性(forward secrecy), 移除了利用静态 RSA 公钥加密算法进行密钥交换的方式, 仅保留 Diffie Hellman(DH)密钥协商算法, 并引入了一个新的密钥协商算法 X25519/448。相比于 TLS 1.3 其他两类 DH 密钥协商算法(有限域 DH 和基于 NIST-P 曲线的椭圆曲线 DH), X25519/448 的计算量更小且参数的选取过程公开, 更受产业界青睐。事实上, 包括 OpenSSH 在内的众多开源项目已经将 X25519/448 作为默认的密钥协商算法。虽然 X25519/448 的计算量相对较小, 但是在云计算、电子交易等大规模并发请求的场景下, 它所依赖的椭圆曲线点乘运算仍然是性能瓶颈。本文利用图形处理器(Graphics Processing Unit, GPU)针对 X25519/448 进行了多层次的性能优化, 同时考虑了可能的计时攻击威胁, 完成性能的最大化。所实现的 X25519/448 在桌面级 GPU GTX 1080 达到每秒 2860412/357944 次操作, 在嵌入式 GPU Tegra X2 上达到每秒 155459/17909 次操作, 性能远远超过 CPU、FPGA 和同类 GPU 平台实现。其中, Tegra X2 上的 X25519 实现分别是 ARM CPU 的 8.5 倍和 FPGA 的 13.2 倍, 体现了 GPU 在嵌入式密码计算领域的强大潜能。

关键词 Curve25519; Curve448; GPU; 密钥协商
中图法分类号 TP309.2 DOI 号 10.19363/J.cnki.cn10-1380/tn.2020.11.06

Implementing High-performance X25519/448 Key Agreement Scheme in General Purpose GPUs

DONG Jiankuo[1,2,3], ZHENG Fangyu[1,2], LIN Jingqiang[1,2,3]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2] Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing 100093, China
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract Widely used in a large range of Internet security protocols such as TLS/SSL, the key exchange provides a method to establish a shared secret between two parties in unprotected channel. Especially in TLS 1.3, in order to ensure forward secrecy, the key exchange method using static RSA public key encryption algorithm is removed, only Diffie Hellman (DH) key agreement algorithm is retained, and a new key agreement algorithm X25519/448 is introduced. Compared with other two DH key agreement algorithms (finite field DH and elliptic curve DH based on NIST-P curve) in TLS 1.3, X25519/448 has less computation and the process of parameter selection is undisguised, which is more favored by the industry. In fact, many open source projects, including OpenSSH, have adopted X25519/448 as the default key exchange algorithm. Although X25519/448 has relatively small computational complexity, the point multiplication of elliptic curves in large-scale concurrent requests such as cloud computing and electronic transactions, is still a performance bottleneck. In this paper, the Graphics Processing Unit (GPU) is used to optimize the performance of X25519/448 at different levels. And in consideration of the possible threat of timing attack, we maximize the performance. The X25519/448 achieved 2860412/357944 ops/s on the desktop GPU GTX 1080 and 155459/17909 ops/s on the embedded GPU Tegra X2. The performance of the X25519/448 is far outstripped the CPU, FPGA and similar GPU platforms. Among them, the X25519 implementations on Tegra X2 is 8.5 times of ARM CPU and 13.2 times of FPGA, respectively, reflecting the strong potential of GPU in the field of embedded cryptographic computing.

通讯作者: 郑昉昱, 博士, 助理研究员, Email: zhengfangyu@iie.ac.cn。

本论文工作得到自然科学基金“通用计算平台的密钥保护技术研究”(No. 61772518)、自然科学基金“基于并行平台和人工智能加速器的高性能密码计算技术研究”(No. 61902392)和国家重点研发计划网络空间安全重点专项“基于国产密码算法的移动互联网密码服务支撑基础设施关键技术”(No. 2017YFB0802100)资助。
收稿日期: 2018-08-30; 修改日期: 2018-11-20; 定稿日期: 2020-09-22

Key words Curve25519; Curve448; GPU; key agreement

1 引言

当前, 随着计算机在各个领域的广泛应用, 互联网金融、线下支付和线上交易等呈现爆发式增长, 现有的密码计算能力已无法满足这一需求。公钥密码体制是各种安全协议的核心密码技术, 椭圆曲线密码算法[1](Elliptic Curve Cryptography, ECC)作为一种非对称密码算法, 相比于 RSA 算法, 同样安全强度下所需的计算量和密钥长度都要小得多。ECC 在用户的传输安全、隐私保护等领域中有着广泛的应用, 包括密钥协商[2]、数字签名[3]等。虽然相比 RSA, ECC 的计算量已经有了很大的降低, 但是它的数学结构仍较复杂, 在面对海量用户的计算请求时, 椭圆曲线算法相对有限的计算能力仍然是服务器整体性能的主要瓶颈之一。

Diffie-Hellman 密钥协商算法[4]是一种在无保护信道中建立共享密钥的方法, 是众多安全协议的基础。基于椭圆曲线的 Diffie-Hellman(ECDH)密钥协商[5-6]算法由于具有更高的性能, 因而受到业界广泛青睐。据文献[7]报道, 在访问量前 100 的网站中, 72 个网站提供 HTTPS(Hyper Text Transfer Protocol over Secure Socket Layer)连接服务, 其中 69 个站点使用 ECDH 作为密钥协商算法。目前 ECDH 使用最为广泛的曲线是由美国国家标准与技术研究院(National Institute of Standards and Technology, NIST)发布的 NIST-P 椭圆曲线。然而, 在 2013 年“棱镜门”事件中, 前美国中央情报局技术分析员斯诺登披露 Dual_EC_DRBG 随机数存在严重后门; 自此, NIST-P 曲线家族安全性[8]开始受到业界广泛置疑。

2006 年, Curve25519 曲线被 Daniel J. Bernstein[9] 设计并发表; “棱镜门”后, 该曲线受到业界的广泛关注。国际互联网研究专门工作组(Internet Research Task Force, IRTF)发布 RFC 7748[10], 推荐使用 Montgomery 曲线(Curve25519 与 Curve448)作为密钥协商协议, 并分别命名为 X25519 与 X448 密钥协商函数。Curve25519 的参数设计公开, 因此被认为是安全的[8], 同时该曲线具有性能方面的优势, 其性能明显优于 NIST P-256 曲线。目前为止, X25519 已经广泛的应用于各种密码库与各种应用中。2014 年, OpenSSH[11]的密钥协商功能将 Curve25519 作为默认椭圆曲线。同时, 从版本 1.1.0 开始, OpenSSL[12]开始支持 X25519 与 X448 算法。2018 年, IRTF 发布了传输层安全协议 TLS 1.3(RFC 8446[13]), 其中增加 X25519 与 X448 作为密钥协商算法, 增加 Ed25519 与 Ed448 作为签名验签算法。虽然 Curve25519 的性能优于 NIST P-256 曲线, 但实现高速可靠的椭圆曲线密码算法仍然是一项重要而具有挑战性的任务, 迫切需要采用高速计算能力来提高算法性能, 用以支持该密码算法在各个领域快速发展。

当前, 大量的研究工作已经将椭圆曲线密码算法部署在处理器或者协处理器等多种计算平台上, 包括 CPU [12, 14]、Phi[15-16]以及图像处理器(Graphics Processing Unit, GPU)等。其中, 桌面游戏以及人工智能等行业的空前火爆带动 GPU 行业的迅猛发展。伴随着统一计算设备架构(Compute Unified Device Architecture, CUDA)[17]以及开放计算语言(Open Compute Language, OpenCL)[18]的出现, GPU 可作为高性能协处理器承担大量的通用计算任务, 而且在大规模计算领域中, 具有显著的计算优势。Antao 等[19-20]人在剩余数系统(Residue Number System, RNS)基础上, 利用多个线程并行化完成大整数模乘操作, 由于每个字节操作都是独立的, 剩余数系统能够很方便的在 GPU 平台上完成并行化计算。但是, 剩余数系统存在明显的计算缺陷, 完成大整数表示与剩余数表示之间的转换, 需要花费巨大的代价。Bernstein [21]与 Leboeuf [22]等人在 GPU 上利用单个线程实现 Montgomery 乘法, Montgomery 乘法, 作为计算大整数模乘的一种方式, 已经广泛的在 GPU 平台上实现[21, 23-26]。在椭圆曲线的计算中, 由于模数较小, 在可接受的延时范围内, 一般单个线程就可以实现一个完整的椭圆曲线点乘运算。

针对梅森素数域[27], 大整数求模运算可以利用快速约减的方法高效实现。快速约减操作的计算指令量大约为 Montgomery 乘法的方式一半。当前流行的椭圆曲线算法都支持快速约减求模运算, 其中包括 NIST-P 系列曲线以及我国的 SM2 算法[28]。Pan 与 Zheng 等[29-32]相关文献都是利用快速约减方法在梅森素数域[27]上实现大整数模乘运算。2017 年, Pan[29] 在 GPU GTX 780 Ti 上, 将 NIST P-256 验签操作性能提升到 920 kops/s(Kilo Operations Per Second, ops/s), 是已知性能最高的 NIST 曲线未知点标量乘法操作。

当前, 研究人员已经将 Curve25519 曲线在不同的高性能处理器或协处理器平台上实现。Michael 等[33]在多种嵌入式平台上完成 X25519 的密钥协商算法。Philip 等[34]在 FPGA 平台上, 实现了延时约为 92 微秒的 Curve25519 的标量乘法。Armando 等[35]利用

AVX2 指令集实现 ECDH 的密钥协商算法。Mahe 等[36]基于 GTX TITAN、AMD R9 290X 等 GPU 平台, 利用 OpenCL 完成 Curve25519 标量乘法运算。成娟娟等[37]基于 GPU GTX 780 Ti 平台, 实现 X25519 密钥协商函数, 并取得良好的实验效果。

近年来, 英伟达公司发布了多款嵌入式 GPU 平台, 基于该公司的 Kepler/Maxwell/Pascal 等架构核心, 分别命名为 Tegra K1/X1/X2(TK1/TX1/TX2)。嵌入式 GPU 平台具有高效的能效比, 例如嵌入式 GPU 平台 TX2, 拥有两个流处理器簇(Streaming Multiprocessor, SM), 能够提供 0.75 万亿次单精度浮点操作每秒(Tera Floating-point Operations Per Second, TFLOPS)的单精度浮点数能力。目前为止, 包含移动手机(HTC Nexus 9)、平板(小米 MiPad)以及车载中控(奔驰, 英伟达无人驾驶平台)等在内的众多移动设备都使用了英伟达高性能的嵌入式 GPU。

本文基于两种 GPU 平台精心设计并完整实现基于 Curve25519/448 的密钥协商算法函数 X25519/448。该实现方案具有高吞吐、低延时以及高适应性等优势, 而且整体计算延时固定, 具有抵御计时攻击的能力。具体贡献从以下几个方面分别阐述:

- 首先, 本文实现包括快速约减算法在内的高性能 Curve25519/448 大整数域算法; 尤其针对 Curve25519 快速约减算法, 提供一种只需要一轮进位操作的快速约减方案, 既提高了约减的性能, 又可以解决由于进位约减轮数不固定带来的安全威胁, 上述大整数域算法全部使用 CUDA 指令集直接进行优化实现, 能够从底层更加高效的使用 GPU 计算能力。
- 其次, 为提高密钥协商函数 X25519/448 性能, 本文调整点乘算法的实现步骤, 减少大整数算术与临时变量的使用数量, 提高寄存器资源的使用效率, 提升算法的整体性能。
- 最后, 本文针对桌面级 GPU(GTX 1080)和嵌入式 GPU(TX2)两种实验平台分别优化实验, 根据平台特点, 选择不同的实验方案与参数, 达到最佳实验效果。基于相同的 GTX TITAN 实验平台, 本文的性能为文献[36]中 Curve25519 点乘性能的 2.66 倍; 在相同的 GTX 780 Ti 上, 本文性能是已知最快 NIST P-256 点乘(文献[29])吞吐率的 1.71 倍。同时在桌面级 GPU GTX 1080 上, 本文的 X25519/448 的吞吐性能峰值分别达到 2860412/357944 ops/s。基于嵌入式 GPU TX2, 本文 X22519/448 的性能峰值为 155459/17909 ops/s。

2 相关背景介绍

本章节中, 首先介绍 Diffie-Hellman 密钥协商算法以及 ECDH 算法, 然后简单描述椭圆曲线 Curve25519/448 的基本知识, 最后详细介绍 GPU 实验平台 GTX 1080 以及嵌入式 GPU Tegra X2。

2.1 Diffie-Hellman 密钥协商算法与 ECDH

Diffie-Hellman 密钥协商算法用以在公共网络通道中, 通信双方协商对称密钥的一种方式。Diffie-Hellman 密钥协商算法是公钥密码学最早的应用之一。

ECDH 是指将椭圆曲线用作 Diffie-Hellman 密钥协商的算法。ECDH 主要计算流程如图 1 所示:

1) Alice 与 Bob 确定曲线类型 E 与域 $\mathbb{F}p$, 同时在曲线上的基点 $P \in E/\mathbb{F}p$;

2) Alice 生成随机数 k_A, 然后计算点乘结果 $P_A=k_A P$;

3) Bob 生成随机数 k_B, 然后计算点乘结果 $P_B=k_B P$;

4) Alice 与 Bob 交换 P_A 与 P_B;

5) Alice 与 Bob 分别计算 $P_{AB}=k_A P_B=k_B P_A$;

其中 k_A 与 k_B 属于(0, n-1), 其中 n 是由基点 P 生成群的阶。

2.2 Curve25519 与 Curve448

椭圆曲线一般使用 Weierstrass 方程形式进行表示, 型如方程(1)

$$y^2 = x^3 + ax + b \tag{1}$$

1987 年, Montgomery 引入一种新的椭圆曲线, 后被命名为 Montgomery 曲线, 方程如下:

$$By^2 = x^3 + Ax^2 + x \tag{2}$$

Curve25519 是一条 Montgomery 曲线, 在 2006 年首次被 Bernstein[9]提出。Curve25519 建立在 255 比特长度的素域($p = 2^{255} - 19$)上, 能够提供 128 比特的安全强度。同样作为一条 Montgomery 曲线, Curve448[38]的大整数素域为 448 比特长度的 Goldilocks 素域($p = 2^{448} - 2^{224} - 1$), 能够提供 224 比特的安全强度。

$$\text{Curve25519: } y^2 = x^3 + 486662x^2 + x \tag{3}$$

$$\text{Curve448: } \quad y^2 = x^3 + 156326x^2 + x \tag{4}$$

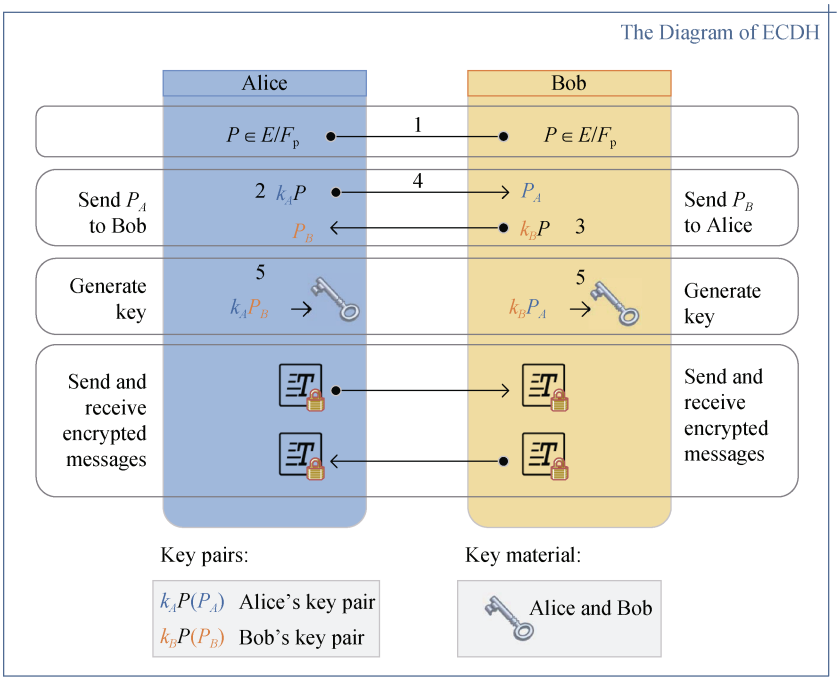

图 1　椭圆曲线 Diffie-Hellman 密钥协商算法流程

Figure 1　The Diagram of ECDH

X25519 与 X448 分别基于上述曲线, 作为椭圆曲线密钥协商算法函数。依据 RFC 7748, X25519/448 两种密钥协商算法为单坐标计算流程, 即计算 kP 的 x 坐标仅根据 P 点的 x 坐标即可完成计算。

2.3　GPU

不同于传统的 CPU 设计, GPU 将更多的晶体管分配给计算单元而非数据缓存与流控制单元, 能够提供更加强大的计算能力。从 2010 年到目前为止, GPU 从 NVIDIA 的 Fermi 架构到 Volta 架构[39], 其计算性能增长已经超过十倍, 相比之下, CPU 性能近十几年来增长十分缓慢。2018 年, 美国超算中心 Summit 超越我国神威-太湖之光, 成为排名世界第一的超算中心, Summit 计算能力主要来自英伟达公司最新的 Volta 架构显卡。

随着 2007 年英伟达发布统一计算设备架构(CUDA),开发者可以借助 CUDA 在性能强大的 GPU 上完成各类通用计算。近年来, 密码研究人员在多种 GPU 平台上完成对密码算法加速, 包括 RSA[25, 40], ECC[29]等。本文的实验平台分为两类, 分别为桌面(desktop)GPU 与嵌入式(embedded)GPU。针对桌面级 GPU, 本文选取市场中最常见的英伟达 GeForce GTX 1080 显卡作为实验平台。它是英伟达于 2016 年发布的基于 Pascal 架构[41]的新一代旗舰显卡, 拥有 2560 个计算核心, 平均分布在 20 个流处理器簇中。GeForce GTX 1080 显卡有 8 GB 显存, 以及具备两个拷贝引擎。嵌入式 GPU TX2 主要硬件信息如图 2 所示, 该嵌入式显卡含有 256 个 Pascal 架构计算核心, 只有一个拷贝引擎, 同时与 CPU 共享 8 GB 内存。在 TX2 中, 每个线程簇拥有 32768 个寄存器, 数量仅为桌面级 GPU 的一半。在两种 GPU 类型中, 存储性能速率从高到低依次为: 寄存器＞片上存储＞片下缓存＞片下存储。

基于 SIMD 的并行计算平台, 英伟达 CUDA 提供了一套 PTX 指令集(Parallel Thread Execution, PTX)[42]。该指令集能够提供可靠稳定的编程指令, 并支持多代 GPU 提供通用计算能力。为方便程序开发, 同时支持嵌入式汇编的开发方式, 开发者可以将 PTX ISA 指令采用内嵌汇编的形式整合到普通的 CUDA 代码中[43], 以提高代码的运行效率。本文中, 我们将代码的核心部分使用 PTX 汇编指令编写, 特别是模乘、模加、模减以及约减等算法实现都是由汇编指令完成。实验中涉及部分 PTX ISA 指令如表 1 所示:

3　X25519/448 密钥协商算法实现

在本章中, 我们完整实现了基于椭圆曲线 Curve25519/448 的密钥协商算法。首先, 本文将详细介绍大整数域算法的实现, 具体包含: 大整数加法、减法、乘(乘加)法、平方; 进而, 通过计算验证分析正确性并最终实现基于 GPU 的快速约减算法; 为提高算法性能, 同时考虑到开发复杂程度, 本文使用 PTX ISA 指令开发上述算法, 并嵌入到 CUDA 代码

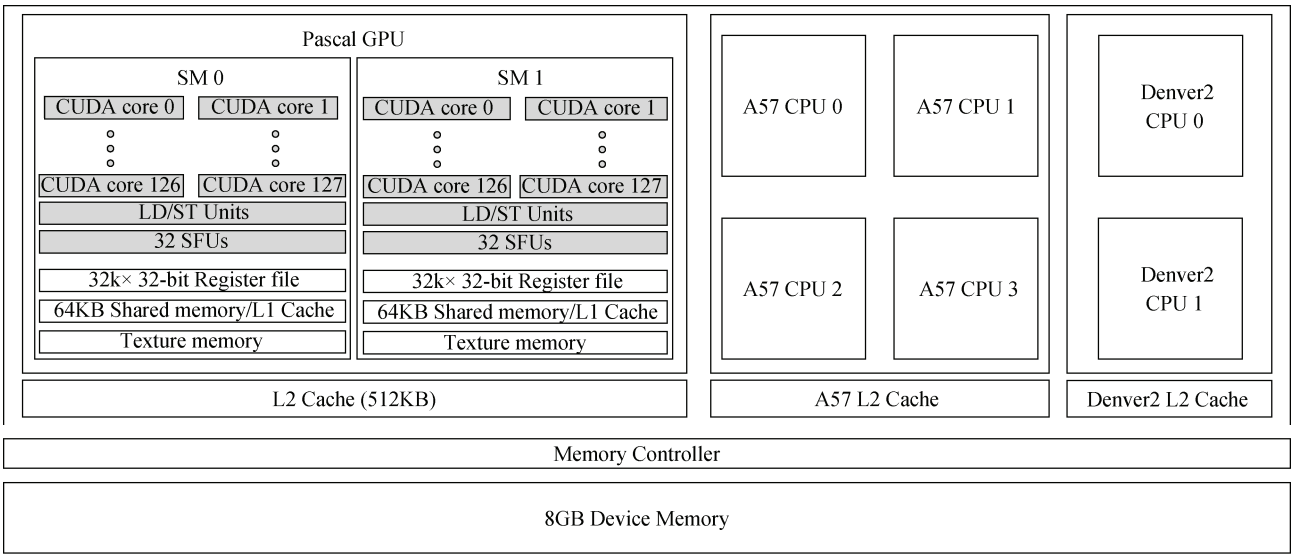

图 2　TX2 主要组成部件结构图

Figure 2　The Components of Tegra X2

表 1　PTX ISA 指令释义

Table 1　PTX ISA Instruction Explanation

类别	具体指令	指令含义
加法	*s=add(a,b)*	*s=a+b*
	s=addc(a,b)	*s=a+b+CF*
减法	*s=sub(a,b)*	*s=a–b*
	s=subc(a,b)	*s=a–b–CF*
乘法	*s=mul.lo(a,b,c)*	$s=(a\times b)_{lo}$
	s=mul.hi(a,b,c)	$s=(a\times b)_{hi}$
乘加	*s=mad.lo(a,b,c)*	$s=(a\times b)_{lo}+c$
	s=mad.hi(a,b,c)	$s=(a\times b)_{lo}+c$
	s=madc.lo(a,b,c)	$s=(a\times b)_{lo}+c+CF$
	s=madc.hi(a,b,c)	$s=(a\times b)_{hi}+c+CF$

(注: 指令后缀.*cc*,产生进位将被写入寄存器 *CC.CF*)

中。最后, 本文通过调整 Montgomery 阶梯算法的实现步骤, 减少临时变量与大整数算术的使用数量, 最终完整实现了高速密钥协商函数 X25519 与 X448。

3.1　大整数算法

由于大整数算法是影响性能核心的部分, 大整数加法与减法的实现相似, 在实现大整数乘法的同时, 本文额外完成大整数乘加算法。此外, 大整数平方计算过程中存在大量重复的执行部分, 本文单独实现大整数平方运算。

3.1.1　加法与减法

大整数加法与减法实现, 基于 CUDA 汇编 PTX ISA 指令集完整实现。如图 3 所示, 我们通过“5-定点数+5-定点数”举例说明加法的实现流程。在 Curve25519 有限域计算中, 本文使用 8 个 *s=add*(*a*,*b*) 指令实现算法, 同样使用 14 加法指令完成 Curve448 大整数加法。需要注意的是, 大整数加法计算的结果包含大整数 *C* 与进位 *carry*, 其中 *carry* 数值不大于 1。约减进位 *carry* 的方法, 我们将在后面的详细描述。大整数减法的实现流程与加法非常相似, 区别点在于使用 PTX ISA 减法指令 *s=sub*(*a*,*b*)。

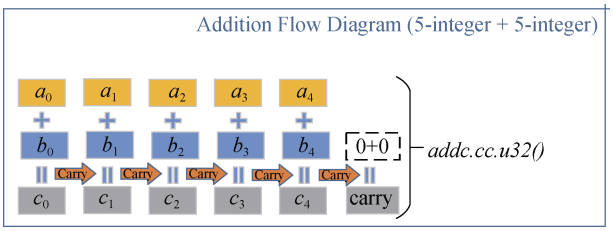

图 3　大整数加法流程图

Figure 3　Large Integer Addition Algorithm

3.1.2　乘法与乘加

大整数乘法主要使用的指令为 *mad.lo* 与 *mad.hi*, 具体的实现流程与文章[30]十分相似。实现的过程中使用的两个基本计算单元为

$$c_{i+j} = madc.lo.u32(a_i, b_j, c_{i+j})$$

$$c_{i+j+1} = madc.hi.u32(a_i, b_j, c_{i+j+1})$$

实验中, 我们同样实现大整数乘加运算 $C=A\times B+D$。相比大整数乘法, 大整数乘加在指令数量保持不变的情况下额外提供一个大整数加法运算。同样, 在图 4 中, 本文提供(5-定点数×5 定点数+5-定点数)算法流程图进行举例说明。

3.1.3　平方

对比于一般的大整数乘法, 可用较少的指令实现大整数平方操作。为提高算法性能, 平方操作被单独实现。具体流程分为三步, 如下:

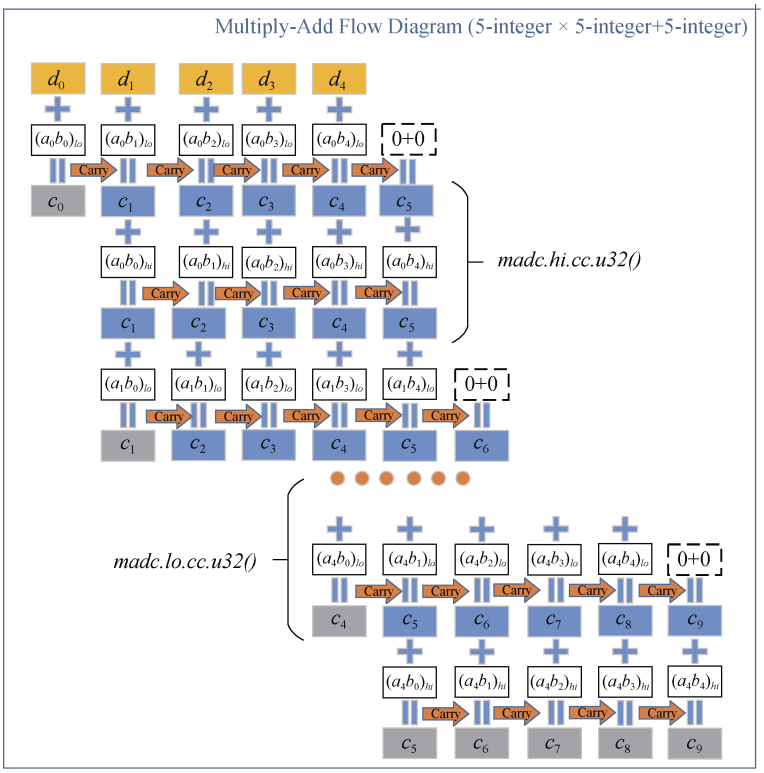

图 4　大整数乘加算法流程图

Figure 4　Large Integer Multiply-Add Algorithm

1) 当($i < j$)时，计算 $a_i \times a_j$，具体与上述大整数乘法相似;

2) 将上述计算结果翻倍，此时，我们已经得到 $a_i \times a_j$ ($i \neq j$)的计算结果;

3) 使用乘加指令 *madc.cc.u32* 将 $a_i \times a_i$ 对应累加到上述计算结果中。

3.2　快速约减算法

本文实现两种曲线的大整数快速约减算法，针对两种不同的曲线，分别提供的不同的约减方案。

3.2.1　Curve25519 快速约减

对于 Curve25519，有限域为 $\mathbb{F}p$，其中 $p = 2^{255} - 19$。GPU 中定点数长度为 32，因此我们使用 8 个 32 比特定点数表示 255 比特长度大整数。因此在大整数乘法中，乘数与被乘数 A, B 的取值范围为$[0, 2^{8\times32})$。由此推断，大整数积 C 范围为$[0,2^{512})$，进一步可以表示为:

$$C \equiv \sum_{i=0}^{15} c_i 2^{32i} \equiv \sum_{i=8}^{15} c_i 2^{32i} + \sum_{i=0}^{7} c_i 2^{32i} \pmod{p} \quad (5)$$

根据 $p = 2^{255} - 19$ 可以得到:

$$2^{255} \equiv 19 \pmod{p} \quad (6)$$

与

$$2^{256} \equiv 38 \pmod{p} \quad (7)$$

此时，方程(5)进一步约减为:

$$\begin{aligned} C &\equiv \sum_{i=0}^{7} 38 c_{i+8} 2^{32i} + \sum_{i=0}^{7} c_i 2^{32i} \pmod{p} \\ &\equiv \sum_{i=0}^{7} (c_i + 38 c_{i+8}) 2^{32i} \pmod{p} \end{aligned} \quad (8)$$

算法 1 Curve25519 大整数乘法结果 512 比特到 256 比特约减流程.

输入: 512 比特长度大整数 $C = \sum_{i=0}^{15} 2^{32i} c_i$，其中 $c_i \in [0, 2^{32})$;

输出: 256 比特长度大整数 $C = \sum_{i=0}^{7} 2^{32i} c_i$，以及 *carry* 其中 $c_i \in [0, 2^{32})$;

1. $carry = 0$
2. FOR i=0 to 3 DO
3. $\quad c_{2i} = madc.lo.cc.u32(c_{2i+8}, 38, c_{2i})$
4. $\quad c_{2i+1} = madc.hi.cc.u32(c_{2i+8}, 38, c_{2i+1})$
5. END FOR
6. $carry = addc.u32(0,0)$
7. FOR i=0 to 2 DO
8. $\quad c_{2i+1} = madc.lo.cc.u32(c_{2i+9}, 38, c_{2i+1})$

9. c_{2i+2} = *madc.hi.cc.u32*(c_{2i+9},38,c_{2i+2})
10. END FOR
11. c_7 = *madc.lo.cc.u32*(c_{15},38,c_7)
12. *carry* = *madc.hi.cc.u32*(c_{15},38,*carry*)

通过方程(8), 512 比特长度大整数乘法结果组成为 16 个 32 比特定点数, 将 38×(c_8,⋯, c_{15})加到(c_0, ⋯, c_7)完成 512 比特约减为 256 比特长度, 同时会产生长度小于 6 比特的进位 *carry*。同时, 在大整数加法的计算结果中, 也存在一个相似 *carry*。根据方程(8)描述, 可以采用(38×*carry* + *C*)的方式进行处理, 但是加法后的计算结果仍有可能产生新的进位。传统的计算方式, 利用 while 循环指令, 不断重复(38×*carry* + *C*)加法, 直到不再产生新的进位为止。但是这种方式存在明显的设计缺陷, GPU 计算过程中, 各线程指令执行不一致时, 导致束分叉(warp divergence), 会造成实验性能大幅度下降。同时, 不同输入产生不同的输出延时, 会带来侧信道攻击的风险。

因此, 在约减进位 *carry* 过程中, 我们必须保证运算时间为定长。针对上述情况, 本文首先提出一种能同时应用于 Curve25519 与 Curve448 有限域的快速约减实现方法。当算法 1 执行完毕时, Curve25519 大整数乘法的结果范围为:

$$(carry_0 \mid C) \in [0, 38\times(2^{256}-1)+(2^{256}-1)] \quad (9)$$

在结果 $carry_0|C$ 中, $carry_0$<38。通过方程(7), Curve25519 约减公式可以表示为:

$$(carry \mid C)+carry\times 38 - carry\times 2^{256} \Rightarrow (carry' \mid C')$$

进一步简化可以得到:

$$C + carry\times 38 \Rightarrow (carry' \mid C') \quad (10)$$

利用公式(10)对 *carry* 进行约减, 为方便描述, 将公式(10)提供的方法称为“Curve25519 进位约减操作”, 第一轮约减操作后, ($carry_1|C_1$)的范围如下所示:

$$\begin{cases} [0, 2^{256}-1] & \text{if } carry_0 = 0 \\ [38, 2^{256}+1405] & \text{if } carry_0 \succ 0 \end{cases} \quad (11)$$

根据上述结果, 可得($carry_1 \mid C_1$)∈[$0, 2^{256}$ + 1405]。此刻, 新产生进位 $carry_1$ 属于{0,1}。再进行第二轮加法约减操作, 可得到($carry_2|C_2$)为:

$$\begin{cases} [0, 2256-1] & \text{if } (carry_1 \mid C_1) \in [0, 2^{256}-1] \\ [38, 1443] & \text{if } (carry_1 \mid C_1) \in [2^{256}, 2^{256}+1405] \end{cases} \quad (12)$$

通过上述两轮“Curve25519 进位约减操作”, 结果($carry_2|C_2$)∈[$0, 2^{256}$–1],可得 $carry_2$ 为 0。计算表明通过两轮约减操作, 可以完全约减进位 *carry*。因此, 实验过程算法结束后, 不考虑 *carry* 的数值, 所有的线程都进行两轮进位约减操作, 用以消除 *carry*。上述方法, 摒弃 while 循环操作, 不会造成线程束分叉, 可以抵御计时攻击, 同时该 Curve25519 进位约减的方式可以推广到 Curve448 曲线。

对于 Curve25519, 本文提供一种更加高效的进位处理方式, 这种方式只需要一轮进位约减操作。此方法使用方程(6)代替方程(7)进行操作。该方案理论上, 将计算结果大整数 *C* 的最高比特位 C_{256} 同样看作进位一部分。其他 255 比特作为 $\widehat{C}$, 约减操作具体应为:

$$(carry_0 \mid C_{256})\times 19 + \widehat{C} \Rightarrow (carry_1 \mid C_1) \quad (13)$$

计算结果 $(carry_1 \mid C_1) \in [0, 2^{255}+1462]$。可得结果小于 2^{256}, 表明进位已经被完全约减。在实验过程中, 单轮进位约减计算方法, 相比于上述两轮约减操作, 能够提高算法整体的计算性能。但是这种方法只能适用于 Curve25519 曲线, 不能应用于 Curve448 曲线。单轮的 Curve25519 进位约减计算步骤参见算法 2。

算法 2. Curve25519 固定时长快速约减流程.

输入: 256 比特大整数 $C = \sum_{i=0}^{7} 2^{32i} c_i$, 其中 $c_i \in [0, 2^{32})$, $carry \in [0, 38]$;

输出: 256 比特大整数 $C = \sum_{i=0}^{7} 2^{32i} c_i$, 其中 $c_i \in [0, 2^{32})$

1. *carry* = *carry*<<1
2. *carry* += c_7 >> 31
3. c_7 &=(1 <<31) – 1
4. c_0 = *mad.lo.cc.u32*(*carry*,19,c_0)
5. FOR i = 1 to 6 DO
6. c_i = *addc.cc.u32*(c_i,0)
7. END FOR
8. c_7 = *addc.u32*(c_7,0)

如图 5 所示, 为 Curve25519 曲线的完整的大整数快速约减流程。对于 Curve25519 大整数模乘操作(包括大整数乘加与平方), 完整的约减流程为步骤(1)到(3)。算法 2 对应图 5 中的步骤(1), 将 512 比特的乘法结果约减为 256 比特。步骤(2)和(3), 主要完成对于算法 1 产生的进位进行约减。同时, 针对 Curve25519 的大整数模加与模减而言, 只需要完成进位的约减操作, 具体对应图 5 中的步骤(2)和(3)。

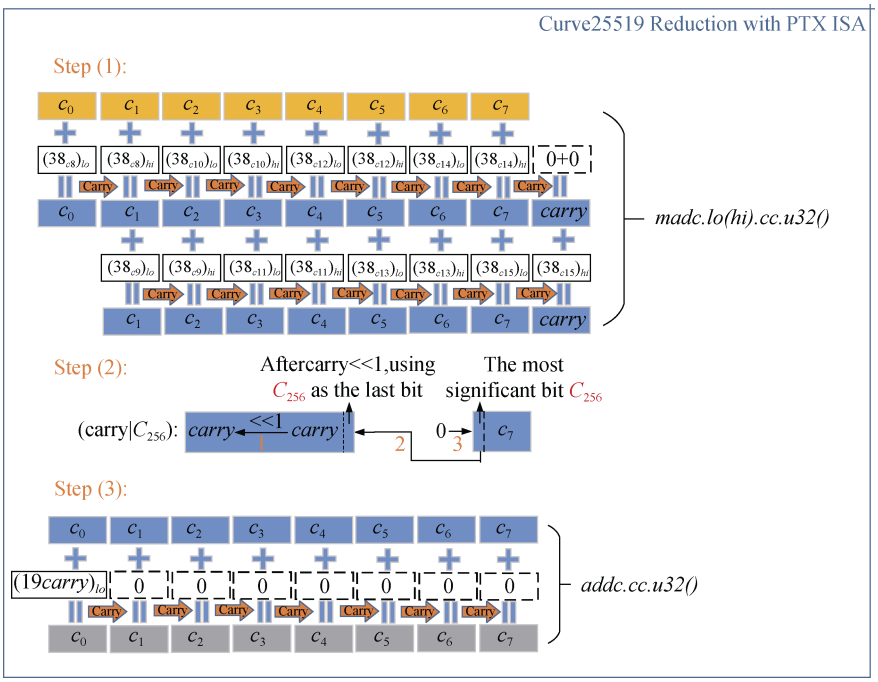

图 5　基于 PTX ISA 指令的 Curve25519 约减流程

Figure 5　Curve25519 Reduction with PTX ISA

3.2.2　Curve448 快速约减

相比于 Curve25519, Curve448 的快速约减更加复杂。大整数乘法 $C=A\times B$ 中, 方程(14)中结果 C 包含 28 个 32 比特的定点数, 共计 896 比特。与 Curve25519 类似, 约减流程首先将 896 比特的 C 约减为 448 比特长度。

$$C\equiv\sum_{i=0}^{27}c_i2^{32i}\equiv\sum_{i=14}^{27}c_i2^{32i}+\sum_{i=0}^{13}c_i2^{32i}\ (\mathrm{mod}\ p)\quad(14)$$

Curve448 中, $p=2^{448}-2^{224}-1$, 可推得:

$$2^{448}\equiv2^{224}+1\ (\mathrm{mod}\ p)\quad(15)$$

通过上述方程(15), 对方程(14)进行转化, 简要的转化过程如下, 利用方程(15)将方程(14)中的高于 2^{448} 的部分 $\sum_{i=14}^{27}c_i2^{32i}$ 转换为:

$$\sum_{i=14}^{27}c_i2^{32i}\equiv\sum_{i=0}^{13}c_{i+14}2^{32i}+\sum_{i=7}^{20}c_{i+7}2^{32i}\quad(16)$$

通过方程(15)对上述结果中高于 2^{448} 部分进一步转化, 整理合并可以得到方程(17):

$$C\equiv\sum_{i=7}^{13}(c_i+c_{i+7}+2c_{i+14})2^{32i}+\sum_{i=0}^{6}(c_i+c_{i+14}+c_{i+21})2^{32i}\ (\mathrm{mod}\ p)\quad(17)$$

根据方程(17)与指令 *addc.cc.u32*, 本文精心设计了算法 3, 将结果 C 从 896 比特长度约减为 448 比特, 同时可能产生额外进位 *carry*, 该进位保持在 2 比特长度内; 在Curve448的加法操作中, 也会产生1比特长度进位。根据方程(17), 进位操作可以依照下述方程进行约减:

$$C\equiv\sum_{i=7}^{13}c_i2^{32i}+carry\times2^{32\times7}+\sum_{i=0}^{6}c_i2^{32i}+carry\ (\mathrm{mod}\ p)\quad(18)$$

算法 3. Curve448 大整数 896 比特到 448 比特约减流程.

输入: 896 比特大整数 $C=\sum_{i=0}^{27}2^{32i}c_i$, 其中 $c_i\in[0,2^{32})$;

输出: 448 比特大整数 $C=\sum_{i=0}^{13}2^{32i}c_i$ 与 *carry*, 其中 $c_i\in[0,2^{32})$;

1. *carry* = 0
2. FOR *i*=0 to 6 DO
3. c_i = *addc.cc.u32*(c_i, c_{i+14})
4. END FOR
5. FOR *i*=7 to 13 DO
6. c_i = *addc.cc.u32*(c_i, c_{i+7})
7. END FOR
8. *carry* = *addc.u32*(0,0)

9. FOR i=0 to 6 DO
10. $c_i = addc.cc.u32(c_i, c_{i+21})$
11. END FOR
12. FOR i=7 to 13 DO
13. $c_i = addc.cc.u32(c_i, c_{i+14})$
14. END FOR
15. $carry = addc.u32(0, carry)$
16. FOR i=7 to 13 DO
17. $c_i = addc.cc.u32(c_i, c_{i+14})$
18. END FOR
19. $carry = addc.u32(0, carry)$

类似的, 我们将算法 4 提供的约减算法流程命名为“Curve448 进位约减操作”。与 Curve25519 对进位处理的第一种方法一致, 同样可以在不考虑进位 *carry* 是否存在情况下, 执行两轮 Curve448 进位约减操作, 保证各个线程指令一致约减进位 *carry*。

算法 4. Curve448 固定时长快速约减流程

输入: 448 比特大整数 $C=\sum_{i=0}^{13}2^{32i}c_i$ 与 *carry*, 其中 $c_i \in [0, 2^{32})$

输出: 448 比特大整数 $C=\sum_{i=0}^{13}2^{32i}c_i$, 其中 $c_i \in [0, 2^{32})$

1. $carry = 0$

Curve448 进位约减:

2. $c_0 = add.cc.u32(c_0, carry)$
3. FOR i=1 to 6 DO
4. $c_i = addc.cc.u32(c_i, 0)$
5. END FOR
6. $c_7 = addc.cc.u32(c_7, carry)$
7. FOR i=8 to 13 DO
8. $c_i = addc.cc.u32(c_i, 0)$
9. END FOR
10. $carry = addc.u32(carry, 0)$

重复执行一轮 Curve448 进位约减

图 6 中, 展示对于 Curve448 完整的约减流程。步骤(1)和(2), 实现了 Curve448 大整数乘法(平方)的积 C 的约减。步骤(1)依据算法 3, 将结果从 896 比特

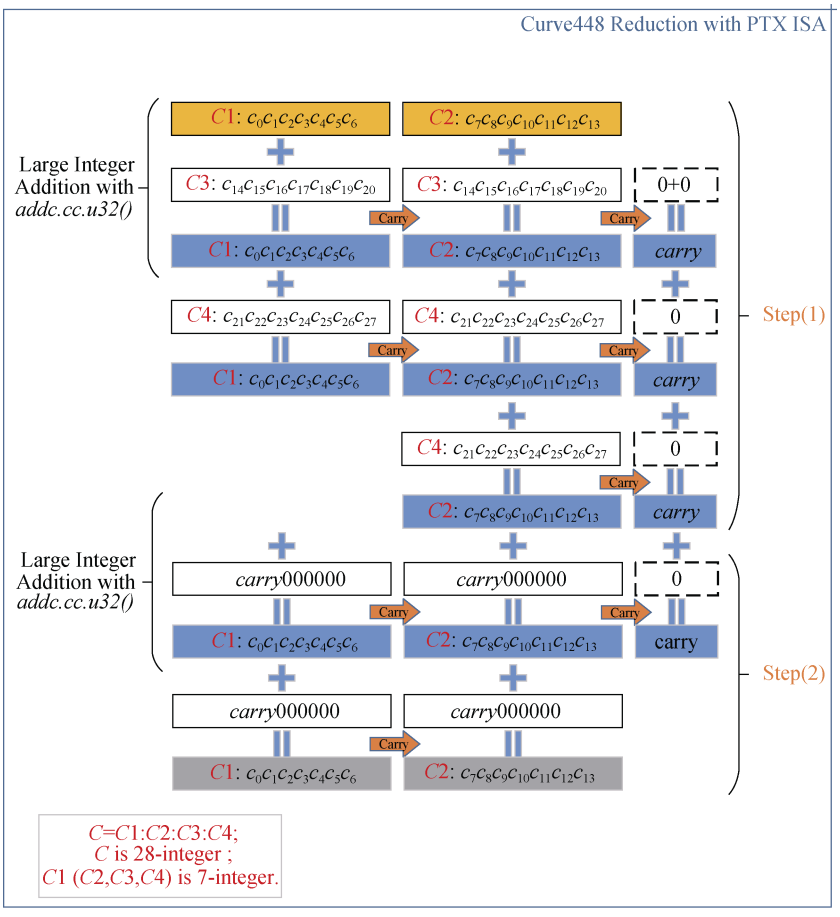

图 6 Curve448 单线程约减流程图

Figure 6 The Curve448 Reduction

约减为 448 比特与进位 *carry*; 步骤(2), 进一步完成进位 *carry* 约减, 按照图 6 方式, 进行两轮累加操作, 可以完全约减进位 *carry*。类似的, 步骤(2)同样支持对于大整数加(减)法进位的约减。总之, Curve448 的快速约减操作中, 各个线程运算指令时间确定, 与输入无关, 具有能够抵御计时攻击的能力。

3.3 椭圆曲线点乘

本文使用 Montgomery 阶梯算法(Montgomery Ladder)实现 X25519 与 X448 椭圆曲线标量乘法。Montgomery 阶梯在计算标量乘法时, 计算时长固定, 与输入无关, 具有抵御计时攻击能力。在 GPU 平台上, 影响 X25519 与 X448 密钥协商算法的主要瓶颈为: 单个线程分配寄存器的数量与有限域算法的执行数量。在不影响算法正确性的前提下, 本文调整了点乘实现过程中有限域算法的执行顺序, 以及尽可能的减少临时变量的使用, 增加变量的重复利用率, 以达到充分利用寄存器资源提高算法性能的目的。算法 5 中, 第 22 步骤, 利用熔加指令 *mad.lo*(*high*)额外实现模乘加算法, 指令数量与模乘算法一致, 功能上相当于一个模乘与一个模加算法, 能够有效减少模加算法的执行数量。该步骤计算过程中, 存在固定数据变量, 用 *a*24 进行表示, 数据 *a*24 被存储到 GPU 的只读内存中, 用以提高访问效率。*a*24 在 X25519/448 具体数值为:

$$X25519: a24 = \frac{(486662-2)}{4} = 121665$$

$$X448: a24 = \frac{(156326-2)}{4} = 39081$$

进一步地, 我们发现, 上述算法的 *a*24 在 Curve25519 与 Curve448 数据均为定值, 若使用二进制表示最多为 17 比特长度, 故在步骤 22 中, 不必实现完整的(256-bit×256-bit+256-bit)乘加算法, 本文在此处的大整数乘加算法 $C=A\times B+D$ 中, 表示 *a24* 的被乘数 B 只有单个定点数 b_0。此时, 省去计算 $A\times(b_1,\cdots,b_7)$对应部分, 得到 $A\times b_0$ 的计算结果, 在使用大整数加法计算 $A\times b_0+D$; 最后, 再使用 Curve25519 的单轮约减算法, 即可得到整个大整数乘加运算计算结果。

算法 5. 基于 Montgomery 阶梯算法的 X25519 与 X448 密钥协商函数.

输入: n 比特长度的标量 k 与随机点 P 的 x-坐标 u

输出: kP 的 x 坐标

1. $x_1=u$; $x_2=1$; $x_3=u$
2. $z_2=0$; $z_3=1$; $swap=1$
3. FOR $i=n-1$ to 0 DO
4. $\quad k_i=(k>>i)\&1$
5. $\quad swap=swap\oplus k_i$
6. (x_2,x_3) = cswap($swap,x_2,x_3$)
7. (z_2,z_3) = cswap($swap,z_2,z_3$)
8. $\quad swap = k_i$
9. $\quad tmp0 = x_3-z_3$
10. $\quad tmp1 = x_2-z_2$
11. $\quad x_2 = x_2-z_2$
12. $\quad z_2 = x_3+z_3$
13. $\quad z_3 = tmp0\times x_2$
14. $\quad z_2 = z_2\times tmp1$
15. $\quad tmp0 = tmp1^2$
16. $\quad tmp1 = x_2^2$
17. $\quad x_3 = z_3+z_2$
18. $\quad z_2 = z_3-z_2$
19. $\quad x_2 = tmp1\times tmp0$
20. $\quad tmp1 = tmp1-tmp0$
21. $\quad z_2 = z_2^2$
22. $\quad tmp0 = tmp1\times(a24+1)+tmp0$
23. $\quad x_3 = x_3^2$
24. $\quad z_3 = x_1\times z_2$
25. $\quad z_2 = tmp1\times tmp0$
26. END FOR
27. (x_2,x_3) = cswap($swap,x_2,x_3$)
28. (z_2,z_3) = cswap($swap,z_2,z_3$)
29. return $x_2\times(z_2^{p-2})$

其中 cswap 为条件互换函数, 实现过程中同样运算时间确定, 本文依据文献[10]提供的方案完成该函数功能。

4 性能评估

本章主要讲述 X25519/448 分别在两种 GPU 平台中的实验结果, 并与其他文献进行性能对比。上述章节介绍了嵌入式 GPU TX2 套件的详细信息, 表 2 中展示桌面级 GPU 实验平台的软件与硬件实验环境。

表 2 桌面级 GPU 平台信息

Table 2 Configuration of Desktop GPU Platform

类别	名称	型号
硬件	CPU	Intel Xeon CPU E5-2690 v2 @ 2.10GHz
	GPU	GeForce GTX 1080
软件	操作系统	Ubuntu 16.04
	工具链	CUDA 8.0

4.1 实现结果

本文完整实现针对 Diffie-Hellman 密钥协商协议的 X25519/448 未知点乘算法。本节主要从两个方面对 GPU 性能进行评价:

- 吞吐性能: 表示 GPU 单位时间内完成 X25519/448 密钥协商函数的数量。
- 延时: 表示 X25519/448 在 GPU 平台上, 从计算请求到计算结束所需要的时间。

同时, 在 GPU 平台实验过程中, 部分参数会影响算法性能, 具体解释如下:

- 线程总数: 为 GPU 内核函数(kernel)开启的线程总数量, 线程块×每块线程数;
- 线程数/线程块: 表示每个线程块中, 启用的线程数量;
- 寄存器量/线程: CUDA 线程中, 可使用的最大寄存器数量。

实验过程中, 针对嵌入式 GPU TX2 平台中, CPU 与 GPU 共享内存的特性, 本文使用了“CUDA 零拷贝内存(Zero Copy)”技术, 与普通的内存拷贝方式的实验结果对比, 性能大约提高 1.5%。

表 3 基于 GPU 平台点乘性能峰值

Table 3 The Peak Performance of Scalar Multiplication on GPU Platforms

算法类别	平台	线程总数	最大寄存器(单线程)	点乘数量	吞吐(ops/s)	延时(ms)
X25519	GTX 1080	20×896	72	20×896	2,860,412	6.26
X448	GTX 1080	20×512	127	20×512	357,944	28.61
X25519	嵌入式 GPU TX2	2×768	40	2×768	155,459	9.88
X448	嵌入式 GPU TX2	2×384	80	2×384	17,909	42.88

由于 X25519 比特长度较小, 较少使用 GPU 寄存器资源, 适合单个线程完成一个点乘整个算法; 对于桌面级 GPU GTX 1080, 本文在点乘请求数量 20×896 取得性能峰值, 为 2860 kops/s, 此时的延时为 6.26 ms。根据图 7, 可以得知, 随着点乘请求数量的增长, X25519 在性能达到峰值之前, 延时变化较小(5.76~6.26 ms); 当请求数量为 20×1024 时, 由于寄存器资源不足, 数据“泄露”到内存中, 导致吞吐性能下降, 延时大大增加。在嵌入式 GPU TX2 中, 由于寄存器资源数量更小, 当内核函数调用线程总数为 2×768, 单个线程最大寄存器量为 40 时, 密钥协商函数 X25519 性能达到最大, 为 155459 次操作每秒, 延时为 9.88 ms。如图 8 所示, 嵌入式 GPU TX2 中, 随着点乘请求数量的增加, 吞吐性能不断增加, 达到峰值后, 同样由于寄存器资源有限, 在点乘请求数量 2×896 以及更大后, 吞吐性能大幅度下降, 延时甚至翻倍。

对于 X448 算法, 比特数较大, 当整个点乘算法用单个线程实现时, 随着请求数量的不断增长, 基于嵌入式 GPU 寄存器平台性能会更早出现资源不足的情况。如图 9, 当密钥协商函数 X448 点乘数量为 20×512 时, 性能便达到峰值 357944 ops/s, 延时为 28.61 ms。同样基于嵌入式 GPU TX2, 请求数量为 2×384 时, 性能达到峰值 17909 ops/s, 延时为 42.88 ms。当吞吐性能达到峰值后, 点乘请求数量的再次增加时, 其吞吐性能快速下降, 延时也会大幅度整张。

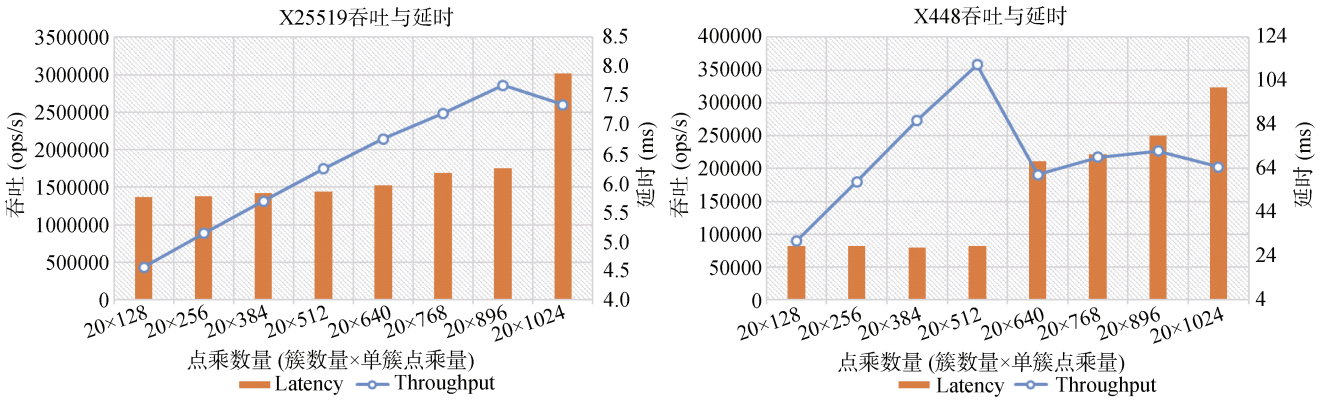

图 7 基于桌面 GPU NVIDIA GTX 1080 的点乘性能随请求数量的变化

Figure 7 The Impact of Batch Size on Performance of Scalar Multiplication with GPU NVIDIA GTX 1080

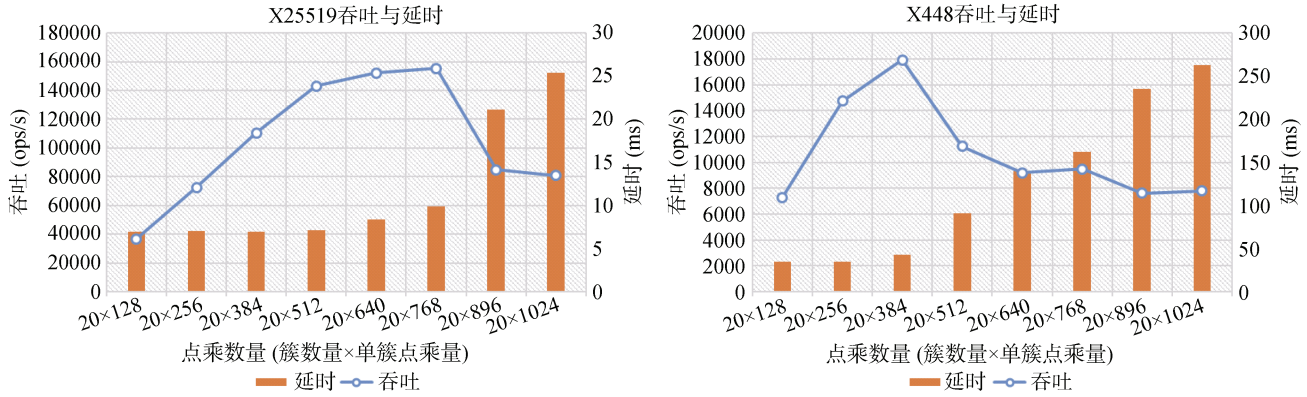

图 8　基于嵌入式 GPU NVIDIA TX2 的点乘性能随请求数量的变化

Figure 8　The Impact of Batch Size on Performance of Scalar Multiplication with GPU NVIDIA TX2

4.2　性能对比

本节将实验成果与其他相关文献进行性能比较, 按照实验平台类型, 分为三类: CPU 平台、FPGA 平台以及 GPU 平台。与平台进行性能分析时, 根据平台特性适当选择本文的桌面级 GPU 或嵌入式 GPU 的实验结果进行对比。当前相关文献主要关注 Curve25519 曲线的应用实现, 对于比特数较长的 Curve448 曲线的研究比较欠缺。

4.2.1　与 CPU 平台进行对比

OpenSSL[12]是一个最常用的安全套接字层密码库, 并提供了 X25519 与 X448 密钥协商算法。在桌面级 GPU 实验平台中, 使用的 CPU 型号为 Intel Xeon E5-2620 V2, 测试点乘过程中, 并行开启 24 线程, 得到 X25519 吞吐性能约为 160 kops/s, X448 吞吐性能为 18.6 kops/s。本文在 GTX 1080 平台上, Curve25519/448 算法性能分别为 OpenSSL 性能的 17.8/19.2 倍。Armando 等[35]使用 AVX2 指令在 Intel Core i7-4770@3.5GHz 平台上完整实现 Curve25519 点乘算法, 具体性能为 156.5×10^3 个时钟周期完成单个点乘计算, 其吞吐性能约为 89.5 kops/s, 远远低于本文在 GTX 1080 平台上实现的性能。

在 Jetson TX2 套件 CPU ARM A57 上运行 OpenSSL 密码算法测试指令 openssl speed ecdhx25519/448, 并开启四个线程测试。从表 4 中可以得到, 本文嵌入式 GPU 上 Curve25519 吞吐性能是同套件 ARM CPU 上 OpenSSL 性能的 8.5 倍, Curve448 吞吐性能为其 OpenSSL 的 4.5 倍。Bernstein 等[44]使用 NEON 指令集, 利用嵌入式 CPU Cortex-A8 单核实现 X25519 算法, 完成单次点乘操作需要 527102 个时钟周期, 吞吐量约为每秒 1517 次点乘操作, 此结果即使推广到四核运算, 性能不到本文嵌入式平台的 1/20。更进一步与服务器级 CPU Intel Xeon E5-2620 v2 性能相比, 其基于 OpenSSL 密码库的性能为 160 kops/s, 而本文中基于功耗仅有 10 瓦左右的嵌入式 GPU TX2 平台, 达到 155.5 kops/s, 性能达到其 97%, 但功耗只有该 CPU 的 1/10。

表 4　相关文献的性能对比

Table 4　Performance Comparison with Related Works

文献	平台	曲线类型	吞吐 (ops/s)
OpenSSL[12]	Quad ARM A57 (4 线程)	Curve25519	18331
		Curve448	3985
	Intel Xeon CPU E5-2620 v2(24 线程)	Curve25519	160803
		Curve448	18632
Bernstein 等[44]	Cortex-A8	Curve25519	1517
Armando 等[35]	Core i7-4770@ 3.5GHz	Curve25519	89456
Philipp 等[34]	FPGA Zynq-7030	Curve25519	10869
Mahe 等[36]	NVIDIA GTX Titan	Curve25519	524288
Pan 等[29]	NVIDIA GTX 780Ti	NIST P-256	929000
成娟娟等[37]	NVIDIA GTX 780Ti	Curve25519	1388400
本文 桌面级 GPU	NVIDIA GTX Titan	Curve25519	1394031
	NVIDIA GTX 780Ti	Curve25519	1588809
	NVIDIA GTX 1080	Curve25519	2860412
	NVIDIA GTX 1080	Curve448	357944
本文 嵌入式 GPU	NVIDIA Tegra X2	Curve25519	155459
		Curve448	17909

与 CPU 平台相比, GPU 的性能优势主要为存在大量的计算核心, 同时, 要达到 GPU 性能峰值, 必须同时存在足够数量的计算请求。GPU 也存在延时较大的缺点, 这是由于 GPU 主频较低造成, 但是在实际应用中, 几毫秒或者二十几毫秒的延时基本不会被用户察觉, 是完全可以接受的。通过上述比较, GPU 平台面对大规模密码运算请求时, 相比于计算能力有限 CPU, 具有较大的性能优势。

4.2.2 与 FPGA 平台进行对比

FPGA 又称现场可编程门阵列, 是一种半定制电路, 能解决定制电路的不足, 同时克服可编程器件门电路数有限的缺点, 被广泛应用于密码运算中。据 Philipp 等[34]描述, 其在 Zynq-7030 平台上实现 Curve25519 点乘延时为 92ms, 吞吐为 10.8 kops/s。该实验结果远远低于本文在 GTX 1080 平台实验结果, 但是功耗方面, GTX 1080 存在较大弱势。进一步, 与嵌入式 GPU TX2 进行比较, 本文中 TX2 中吞吐性能约为 155 kops/s, 性能约为 FPGA 的 14.3 倍。同时与 FPGA 平台相比, 基于 GPU 平台开发密码算法具更高拓展性与适应性。

4.2.3 与 GPU 平台进行对比

目前, 我们还未查阅到基于嵌入式 GPU 上实现非对称密码算法的文献, 因此本文仅与桌面级 GPU 性能进行对比。Mahe 等[36]在平台 GTX Titan, 花费 2.0s 时间完成约 104 万 Curve25519 点乘算法, 其吞吐率为 524 kops/s, 基于相同 GPU 平台, 本文的性能是其结果的 2.56 倍。由于 Mahe 等研究人员使用 OpenCL 对 GPU 进行开发, 不能够高效利用 GPU 计算资源, 本文使用 PTX ISA 指令对大整数乘法算法以及约减部分进行优化加速, 在性能上取得巨大优势。Pan 等[29]在平台 GTX 780Ti 上, 实现 929 kops/s 点乘算法, 是目前已知最快的 NIST 曲线点乘实验结果。基于相同平台 GTX 780Ti, 本文性能达到 1588 kops/s, 为 Pan 等人性能的 1.71 倍, 这是由于本文选择了更加安全高效的 Curve25519 算法以及对该算法实现流程的优化。成娟娟等[37]同样在 GPU GTX 780Ti 平台上实现 X25519 算法, 性能达到 1388400 ops/s, 基于相同的实验平台, 其峰值性能是本文的 87%, 本文的优势主要在于进一步针对 Curve25519 快速约减算法的优化, 以及在点乘算法实现中, 通过调整计算步骤, 减少了大整数算法的数量。

与本文的前期工作[45]进行对比, 本文增加了基于嵌入式 GPU TX2 平台的实现结果。针对嵌入式 GPU 寄存器资源减半以及 CPU 与 GPU 共享内存实际情况, 本文通过减少内核函数的并行度, 使用“零拷贝”内存技术等优化方案, 在大量的实验结果中, 得到最优结果的实验参数。相比于桌面级 GPU, 嵌入式 GPU 具有更高的实验能效比。完整的桌面级 GPU 实验平台功耗约为 400W (GPU 250W + CPU 150W), 而嵌入式 GPU TX2 功耗约为 10W, 通过计算可知, 单位功耗的情况下, 基于嵌入式 GPU TX2 实验平台的 X25519/448 性能约为桌面级 GPU 1080 的 2 倍, 具有显著的能效比优势。

通过对比结果发现, 基于相同的 GPU 平台, 相比其他文献, 本文 Curve25519 点乘实验结果具有巨大的性能优势, 并创造了新的性能记录。目前为止, 还未发现文献报道Curve448的在GPU平台上点乘性能结果。

5 总结

在 TLS 1.3 被逐步推广的大背景下, 拥有更高性能及安全特性的 Curve25519/448 曲线, 必将成为学者以及工业界的研究热点。本文中, 使用 CUDA PTX ISA 指令,分别基于两种类型图形处理器完整实现 X25519/448 密钥协商函数, 在兼顾安全实现的基础上, 取得显著的性能优势: 所实现的 X25519/448 在桌面级 GPU GTX 1080 达到每秒 2,860,412/357,944 次, 在嵌入式 GPU Tegra X2 上达到每秒 155459/17909 次, 性能远远超过 CPU、FPGA 和同类 GPU 平台实现。

未来, 我们将关注多线程拆分实现的低延时密钥协商算法, 以及安全且高性能的签名验签算法,比如 EdDSA(Edwards-curve Digital Signature Algorithm)等。

致 谢 感谢自然科学基金“通用计算平台的密钥保护技术研究”(No.61772518)、“基于并行平台和人工智能加速器的高性能密码计算技术研究”(No.61902392)和国家重点研发计划网络空间安全重点专项“基于国产密码算法的移动互联网密码服务支撑基础设施关键技术”(No.2017YFB0802100)对本文的资助。

本文的早期版本“Towards High-performance X25519/448 Key Agreement Scheme in General Purpose GPUs”发表于国际会议 2018 *IEEE Conference on Communications and Network Security(CNS)*。

参考文献

[1] Lochter M, Merkle J. Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation[R]. RFC Editor, 2010.

[2] Merkle R C. Secure Communications over Insecure Channels[J]. *Communications of the ACM*, 1978, 21(4): 294-299.

[3] Boneh D. Digital Signature Standard[J]. *TELECOMMUNICATION ENGINEERING*, 1995:158-159.

[4] Mahalanobis A. Diffie-hellman key exchange protocol, its generalization and nilpotent groups[M]. Florida Atlantic University, 2005.

[5] Koblitz N. Elliptic Curve Cryptosystems[J]. *Mathematics of Com-*

putation, 1987, 48(177):203-209.

[6] Miller V S . Use of Elliptic Curves in Cryptography[C]. *Lecture notes in computer sciences 218 on Advances in cryptology---CRYPTO 85*. Springer Berlin Heidelberg, 1986.

[7] Harkanson R, Kim Y. Applications of Elliptic Curve Cryptography: A Light Introduction to Elliptic Curves and a Survey of Their Applications[C]. *the 12th Annual Conference on Cyber and Information Security Research*, 2017: 265-271.

[8] SafeCurves: choosing safe curves for elliptic-curve cryptography. https://safecurves.cr.yp.to/. Jun. 2014.

[9] Bernstein D J. Curve25519: New Diffie-Hellman Speed Records[M]. Public Key Cryptography - PKC 2006. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006: 207-228.

[10] Langley A, Hamburg M, Turner S. Elliptic Curves for Security[R]. RFC Editor, 2016.

[11] Openssh, http://www.openssh.com/index.html, 2017.

[12] OpenSSL: Cryptography and SSL/TLS toolkit.. https://www.openssl.org. Sept. 2020.

[13] Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3[R]. RFC Editor, 2018.

[14] Zhao Y, Lin J Q, Pan W Q, et al. RegRSA: Using Registers as Buffers to Resist Memory Disclosure Attacks[J]. *ICT Systems Security and Privacy Protection*, 2016: 293-307.

[15] Shun Y, Dantong Y. PhiOpenSSL: Using the Xeon Phi Coprocessor for Efficient Cryptographic Calculations[C]. *IEEE International Parallel and Distributed Processing Symposium*, 2017:565-574.

[16] Zhao Y, Pan W Q, Lin J Q, et al. PhiRSA: Exploiting the Computing Power of Vector Instructions on Intel Xeon Phi for RSA[C]. *International Conference on Selected Areas in Cryptography*, 2016: 482-500.

[17] CUDA C++ Programming Guide, https://docs.nvidia.com/cuda/cuda-c-programming-guide/. 2017.

[18] Stone J E, Gohara D, Shi G C. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems[J]. *Computing in Science & Engineering*, 2010, 12(3): 66-73.

[19] Antao S , Bajard J C , Sousa L . Elliptic Curve point multiplication on GPUs[C]. *IEEE International Conference on Application-specific Systems Architectures & Processors*, 2010: 192-199

[20] Antao S, Bajard J C, Sousa L. RNS-Based Elliptic Curve Point Multiplication for Massive Parallel Architectures[J]. *The Computer Journal*, 2012, 55(5): 629-647.

[21] Bernstein D J, Chen H C, Chen M S, et al. The billion-mulmod-per-second PC[C]. *Workshop record of SHARCS*. 2009: 131-144.

[22] Leboeuf Karl, Muscedere R, Ahmadi Mahdi. A GPU implementation of the Montgomery multiplication algorithm for elliptic curve cryptography[C]. *2013 IEEE International Symposium on Circuits and Systems*, 2013: 2593-2596.

[23] Bernstein D J, Chen T R, Cheng C M, et al. ECM on Graphics Cards[M]. Advances in Cryptology - EUROCRYPT 2009. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009: 483-501.

[24] Harrison O, Waldron J. Efficient Acceleration of Asymmetric Cryptography on Graphics Hardware[C]. *Progress in Cryptology - AFRICACRYPT*, 2009: 350-367.

[25] Jang K, Han S, Han S, et al. SSLShader: Cheap SSL Acceleration with Commodity Processors[C]. *NSDI*. 2011: 1-14.

[26] Code C. Fast GPU Based Modular Multiplication. https://on- demand.gputechconf.com/gtc/2014/poster/pdf/P4156_montgomery_multiplication_CUDA_concurrent.pdf.

[27] Solinas J A. Generalized Mersenne numbers[M]. Faculty of Mathematics, University of Waterloo, 1999.

[28] Man-gui L I. Study on Public Key Infrastructure in Support of Public Key Cryptographic Algorithm SM2 based on Elliptic Curves [J]. *Information Security and Communications Privacy*, 2011, 9: 78-80.

[29] Pan W Q, Zheng F Y, Zhao Y, et al. An Efficient Elliptic Curve Cryptography Signature Server with GPU Acceleration[J]. *IEEE Transactions on Information Forensics and Security*, 2017, 12(1): 111-122.

[30] Zheng F, Pan W, Lin J, et al. Exploiting the potential of GPUs for modular multiplication in ECC[C]. *International Workshop on Information Security Applications*. 2014: 295-306.

[31] Bos J W. Low-Latency Elliptic Curve Scalar Multiplication[J]. *International Journal of Parallel Programming*, 2012, 40(5): 532-550.

[32] Szerwinski R, Güneysu T. Exploiting the Power of GPUs for Asymmetric Cryptography[M]. Cryptographic Hardware and Embedded Systems – CHES 2008. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008: 79-99.

[33] Düll M, Haase B, Hinterwälder G, et al. High-speed Curve25519 on 8-bit, 16-bit, and 32-bit Microcontrollers[J]. *Designs, Codes and Cryptography*, 2015, 77(2/3): 493-514.

[34] Koppermann P, de Santis F, Heyszl J, et al. Low-latency X25519 Hardware Implementation: Breaking the 100 Microseconds Barrier[J]. *Microprocessors and Microsystems*, 2017, 52: 491-497.

[35] Faz-Hernández A, López J. Fast Implementation of Curve25519 Using AVX2[M]. Progress in Cryptology -- LATINCRYPT 2015. Cham: Springer International Publishing, 2015: 329-345.

[36] Mahé, J.M. Chauvet. Fast GPGPU-Based Elliptic Curve Scalar Multiplication[EB/OL]. *IACR Cryptology ePrint Archive,* 2014: 198.

[37] 成娟娟, 郑昉昱, et al. Curve25519 椭圆曲线算法 GPU 高速实现[C]. *第32 次全国计算机安全学术交流会论文集*. 2017: 122-127.

[38] Hamburg M. Ed448-Goldilocks, a new elliptic curve[EB/OL].

IACR Cryptology ePrint Archive, 2015: 625.

[39] Nvidia tesla v100 GPU architecture. https://images nvidia. com/content/volta-architecture/pdf/volta-architecture-whitepaper. pdf. 2017.

[40] Dong J K, Zheng F Y, Pan W Q, et al. Utilizing the Double-Precision Floating-Point Computing Power of GPUs for RSA Acceleration[J]. *Security and Communication Networks*, 2017, 2017: 1-15.

[41] Nvidia Corporation. Nvidia Tesla P100 whitepaper. https://www. nvidia.cn/data-center/resources/pascal-democratization-whitepaper/, April, 2016.

[42] Nvidia Corporation. Parallel Thread Execution ISA Version 6.0 https://docs.nvidia.com/cuda/archive/9.0/pdf/ptx_isa_6.0.pdf. Jun, 2017.

[43] NVIDIA C. Nvidia tesla v100 gpu architecture. https://images. nvidia. com/content/volta-architecture/pdf/volta-architecture-whitepaper. pdf. Aug, 2017.

[44] Bernstein D J, Schwabe P. NEON crypto[C]. *International Workshop on Cryptographic Hardware and Embedded Systems. Springer*, 2012: 320-339.

[45] Dong J, Zheng F, Cheng J, et al. Towards High-performance X25519/448 Key Agreement in General Purpose GPUs[C]. *2018 IEEE Conference on Communications and Network Security (CNS)*, 2018: 1-9.

董建阔 于 2014 年于西安交通大学计算机科学与技术专业获得工学学士学位。现中国科学院信息工程研究所网络空间安全专业攻读工学博士学位。研究领域为基于 GPU 的非对称密码算法安全高速实现。Email: dongjiankuo@iie.ac.cn

林璟锵 于 2009 年在中国科学院研究生院信息安全专业获得博士学位。现任中国科学院信息工程研究所研究员。研究领域为数据安全与隐私、应用密码学, 尤其是密码技术在计算机网络系统的应用。Email: linjingqiang@iie.ac.cn

郑昉昱 于 2011 年在中国科学技术大学获得学士学位, 2016 年在中国科学院大学信息安全专业获得博士学位。现任中国科学院信息工程研究所助理研究员。研究领域为应用密码学、高性能计算和计算机算术。Email: zhengfangyu@iie.ac.cn