V-Curve25519: Efficient Implementation of Curve25519 on RISC-V Architecture

Qingguan Gao[1], Kaisheng Sun[2], Jiankuo Dong[2(✉)], Fangyu Zheng[3], Jingqiang Lin[4], Yongjun Ren[5], and Zhe Liu[6]

[1] School of Cyber Science and Engineering, Southeast University, Nanjing, China
[2] School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China
djiankuo@njupt.edu.cn
[3] School of Cryptology, University of Chinese Academy of Sciences, Beijing, China
[4] School of Cyber Security, University of Science and Technology of China, Hefei, China
[5] Hangzhou Post Quantum Cryptography Technology Co., Ltd., Hangzhou, China
[6] Nanjing University of Aeronautics and Astronautics, Nanjing, China

Abstract. Internet of Everything technology has greatly promoted the development of intelligent Internet of Vehicles (IoV) system. Similar to the Internet of Things system, the Internet of Vehicles also faces the problems of shortage of computing resources and weak security protection. Open-source RISC V is an important solution for Cloud-to-Edge collaborative SoC chips in Vehicle Networking System. Research on RISC-V based cryptography, especially public key cryptography with high computational complexity, can provide efficient cryptographic support for security authentication, signature generation, data encryption and so on. In this paper, based on the RISC-V 64-bit instruction set, we propose several methods to improve the performance of Curve25519 public key cryptography algorithm, abbreviated as V-Curve25519. V-Curve25519 optimizes the implementation of Curve25519 cryptography from large integer representation, finite field, point arithmetic and scalar multiplication, in which the large integer operation optimizations can be extended to other elliptic curve public key cryptography schemes. Our V-Cruve25519 also takes into account the side-channel protection security implementation, which ultimately meets the constant-time computing latency. On the same platform, the proposed V-Curve25519 improves by 35% compared to the state-of-the-art Curve25519 implementation.

Keywords: RISC-V · ECC · Curve25519 · Public Key Cryptography

This work was supported in part by the National Key R & D Program of China under Grant No. 2022YFB2701400, in part by Major Science and Technology Demonstration Project of Jiangsu Provincial Key R & D Program under Grant No. BE2022798, in part by the National Natural Science Foundation of China under Grant No. 62302238, in part by the Natural Science Foundation of Jiangsu Province under Grant No. BK20220388, in part by the Natural Science Research Project of Colleges and Universities in Jiangsu Province under Grant No. 22KJB520004, in part by the China Postdoctoral Science Foundation under Grant No. 2022M711689, in part by National Cryptography Development Fund No. MMJJ20180105.

C. Ge and M. Yung (Eds.): Inscrypt 2023, LNCS 14527, pp. 130–149, 2024.
https://doi.org/10.1007/978-981-97-0945-8_8

1 Introduction

With the rapid development of science and technology, the Internet of Things (IoT) technology has been widely used in all walks of life and infiltrated into all aspects of human life. The concept of IoT firstly appeared in the book "The Road Ahead" written by Bill Gates in 1995 [1]. It is the third information technology revolution after PC and Internet. It is a network that is expanded and extended on the basis of Internet. By combining various sensor devices with networks, the IoT forms a huge network to realize the interconnection between the physical world and the information world. As an important part of the new generation of information technology, the application fields of IoT technology have involved many aspects such as industry, agriculture, environment, medical treatment, transportation, and home, effectively promoting the intelligent development of all walks of life and greatly improving the quality of people's life. In recent years, the number of IoT devices has also shown explosive growth, and countries have invested huge costs in research and development of IoT technology.

Internet of Vehicles (IoV) has become an important intersection of the two areas of strategic emerging industries, the IoT and intelligent automobiles. The so-called IoV refers to the extraction and effective utilization of static and dynamic information of all vehicles on the information network platform through wireless technology through electronic devices loaded on vehicles. However, as shown in Fig. 1, compared with traditional IoT, the architecture of IoV is usually more complex, and because the physical location of the vehicle carrier itself is often in a high-speed moving state, while identity authentication has problems such as high communication costs, high computing costs, and low adaptability under IoV, so security issues become an urgent problem to be solved in IoV technology. It is very important to ensure personal information security, enterprise information security, and even national information security in the application of IoV technology, and to achieve a balance between informatization and security.

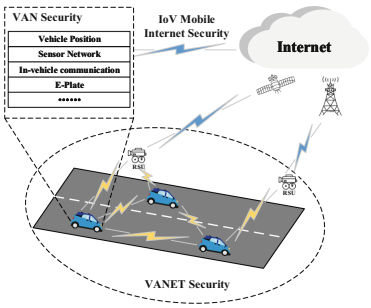

Fig. 1. The Architecture of IoV Security

Public key cryptography was born in 1976 [2], also known as asymmetric key cryptography. It is the core of modern cryptography and is widely used in many

fields, such as public key encryption and decryption, key negotiation [3], digital signature [4], and secure multi-party computation [5]. Unlike symmetric cryptography, which uses the same key to perform encryption and decryption operations, public key cryptography divides the key into two parts: the public key and the private key. The public key is used for encryption and can be made public, while the private key is used for decryption and can only be held by the user. It needs to be kept strictly confidential. Although public key cryptography breaks the restriction that the encryption and decryption keys of traditional symmetric cryptography must be the same, the cost is that the key length and computational complexity are greatly increased compared with symmetric cryptography [6]. Therefore, it has an unbearable performance bottleneck when applied to the IoV devices with insufficient storage resources and computing resources. The security of public key cryptography originates from the mathematical difficulties used at the bottom. For example, the RSA algorithm officially released in 1978 is implemented based on the difficult problem of large integer decomposition. Due to the rapid improvement of computer performance in recent years, the key length of RSA mainstream must be more than 1024 bits [7], otherwise, there is a risk of exhaustive attack. The elliptic curve cryptosystem [8] proposed in 1985 is based on the difficulty of elliptic curve discrete logarithm problem. Compared with RSA, it has higher security, shorter key length, and better flexibility. It is very suitable for the IoT devices with limited resources and has attracted extensive attention from researchers.

Curve25519 is a Montgomery curve and one of the fastest elliptic curves at present. It was designed and published by Professor Daniel J. Bernstein in 2006 [9]. After Curve25519 was published, it received extensive attention from academic and industrial circles. Since the design parameters of Curve25519 are public, it is considered safe. On the other hand, due to its characteristics, Curve25519 has obvious performance advantages over NIST p-256 curve. In January 2018, the Internet Research Task Force (IRTF) released RFC 7748 [10], taking Curve25519 as the recommended curve of elliptic curve key negotiation protocol and naming it X25519 key negotiation function. When communicating on an insecure channel, through X25519, the communication parties can select a 32-byte private key by themselves, and calculate their public key through curve255119 scalar multiplication. The communication parties can calculate a 32-byte key shared by both parties through their private key and the other party's public key, which can be used for subsequent identity authentication and message encryption. Since the release of Curve25519, it has been widely used in various password libraries and security services. Starting from version 1.1.0, OpenSSL [11] supports the X25519 algorithm. In 2018, IRTF issued RFC 8446 [12], in which X25519 was used as the key negotiation algorithm of transport layer security (TLS) protocol, and ed25519 was added as the signature verification algorithm. Although Curve25519 is currently one of the fastest elliptic curves, its computational complexity is still too high to a certain extent, especially when used in some low-performance embedded devices. Therefore, there is an urgent need to optimize the Curve25519 calculation process from bottom to top to improve the overall performance, so

as to drive the rapid development of this cryptographic algorithm in the field of IoT.

RISC-V is a free and open-source Instruction Set Architecture (ISA), which was initiated by University of California in 2010. RISC-V was founded as the RISC-V Foundation from 2015 and is incorporated today as RISC-V International Association [13]. Up to now, RISC-V can support 32-bit, 64 bit and even 128bit versions of secure microprocessors.

1.1 Contributions and Paper Organization

In this paper, based on 64-bit RISC-V platform, the key negotiation algorithm function X25519 based on Curve25519 is fully implemented, and the underlying finite field operation of Curve25519 scalar multiplication is carefully designed, including modulo addition, modulo subtraction, modulo multiplication, modulo square and modulo reduction, which greatly improves its operational efficiency. The implementation scheme (short for V-Cruve25519) has the advantages of high throughput, low delay, and high availability compared with existing works. The specific contributions are described in the following aspects:

- Firstly, based on RISC-V instruction set architecture, this paper fully implements the key negotiation function X25519 of efficient elliptic curve cryptography Curve25519. Based on the problem of insufficient authentication performance prevalent in the current IoV system, we select the RISC-V embedded development board VisionFive as the encryption computing accelerator platform, in order to improve the computing efficiency of message encryption and identity authentication in the IoV system. Compared with other related work, the various optimization methods in this paper can significantly improve the performance of scalar multiplication of elliptic curves. In addition, the work in this paper is not limited to Curve25519, which can also be used in other elliptic curve cryptography algorithm, thus bringing some performance improvement.
- Secondly, based on the 64-bit RISC-V architecture core, this paper optimizes the finite field layer operation of the elliptic curve Curve25519 to fully exploit the word length advantage of a 64-bit processor. To our knowledge, this paper is the first to fully implement X25519 key negotiation algorithm based on the 64-bit RISC-V kernel, which fills in the gap of insufficient optimization of related algorithms on the 64-bit RISC-V processor. In addition, this paper uses a constant computation delay scheme for the finite field large integer reduction method and uses the Montgomery Ladder algorithm with the same fixed delay to calculate the elliptic curve scalar multiplication, which makes the algorithm to some extent resistant to side-channel attacks.
- Finally, through the above optimization methods, we have mined the scalar multiplication performance of Curve25519 on the RISC-V architecture from the bottom to the top, reaching the latest performance record of 64-bit RISC-V processors. At 15W power consumption, our experimental device VisionFive can provide scalar elliptic curve multiplication of 3378 ops/s with a delay of 0.296 ms, which is 1.35 times the implementation of OpenSSL on the same

platform. Our optimized scheme provides a more efficient choice for security in IoV devices.

The rest of this paper is organized as follows:

Section 2 explains some preliminaries including Curve25519, ECDH, RISC-V ISA, and our experimental platform, VisionFive development board. Section 3 describes the proposed strategies for the key negotiation algorithm function X25519 implementation. Section 4 performs our optimized implementation and compares it with related works. Section 5 concludes the paper and looks forward to future works.

2 Preliminaries

In this chapter, first, the basic knowledge of elliptic curve Curve25519 is briefly described, then the Diffie-Hellman key negotiation algorithm and ECDH algorithm are introduced. Finally, the RISC-V instruction set architecture and the embedded development board StarFive VisionFive based on RISC-V 64-bit processor are introduced in detail.

2.1 Curve25519

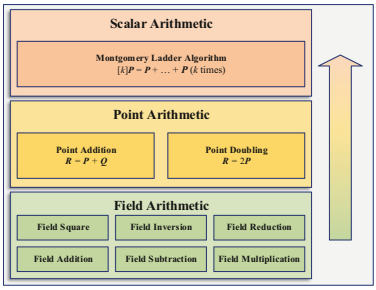

Fig. 2. The Architecture of ECC

As shown in Fig. 2, the architecture of ECC can be divided into finite field arithmetic layer, point arithmetic layer, and scalar multiplication arithmetic layer from bottom to top. Among them, the calculation of the upper layer needs to call the calculation of the lower layer to complete, so optimizing the calculation process at the arithmetic level of the lower finite field of the elliptic curve is the core of improving the efficiency of the elliptic curve cryptosystem.

Generally, the basic concept and operation principle of elliptic curve are introduced by taking Weierstrass curve as an example, because any elliptic curve can be written in the form of Weierstrass curve. In fact, elliptic curves also

include many other types, such as Montgomery curve, twisted Edwards curve, and so on.

Weierstrass elliptic curve equation is as follows:

$$E/\mathbb{F}_p : y^2 = x^3 + Ax + B, \tag{1}$$

where $A, B \in \mathbb{F}_p$, $4A^3 + 27B^2 \neq 0$.

Montgomery elliptic curve equation is as follows:

$$E/\mathbb{F}_p : By^2 = x^3 + Ax^2 + x, \tag{2}$$

where $A, B \in \mathbb{F}_p$, $B(A^2 - 4) \neq 0$.

Edwards elliptic curve equation is as follows:

$$E/\mathbb{F}_p : ax^2 + y^2 = 1 + dx^2y^2, \tag{3}$$

where $a, d \neq 0$, $a \neq d$.

Curve25519 was proposed by the famous cryptographer Daniel J. Bernstein [9]. It is an elliptic curve defined on the Montgomery curve, where $p = 2^{255} - 19$, $A = 486662$, $B = 1$, base point is defined as:

$$(9, 14781619447589544791020593568409986887264606134616475288964881837755586237401) \tag{4}$$

2.2 Diffie-Hellman Key Negotiation Algorithm and ECDH

Diffie-Hellman key negotiation algorithm was first proposed by Diffie and Hellman in an original paper in 1976 [14]. It is used by communication parties to negotiate a symmetric key in an insecure channel so that messages can be encrypted using the key in subsequent communication. It has been widely used in many security products.

ECDH applies elliptic curve to DH key negotiation and exchanges the keys of communication parties through the mathematical problem of elliptic curve discrete logarithm. The main calculation flow of ECDH is shown in Fig. 3:

1. Alice and Bob determine the curve type E, the finite field $\mathbb{F}_p$, and the base point G on the elliptic curve. n is the order of the base point G, that is, the minimum positive integer satisfying $nG = 0$;
2. Alice generates a random number $k_A \in (0, n-1)$ and calculates the scalar multiplication of elliptic curve $P_A = k_A G$;
3. Bob generates a random number $k_B \in (0, n-1)$ and calculates the scalar multiplication of elliptic curve $P_B = k_B G$;
4. Alice and Bob exchange P_A and P_B;
5. Alice and Bob respectively calculate $K_{share} = k_A P_B = k_B P_A$.

Since then, Alice and Bob have negotiated the key K_{share} only known to them and can communicate securely on the insecure channel through symmetric cryptography.

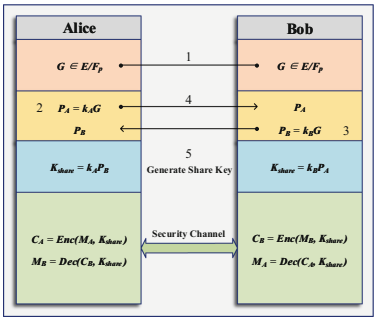

Fig. 3. The Diagram of ECDH

Table 1. Composition of RISC-V Basic ISA

Name	Numbers	Description
RV32I	47	Integer instruction, including arithmetic, branch, and memory access. 32 bits address space, 32-bit registers
RV32E	47	The instruction is the same as RV32I, but the number of registers is changed to 16, which is used in embedded environment
RV64I	59	Integer instruction, 64 bits addressing space, 32 64-bit registers
RV128I	71	Integer instruction, 128 bits address space, 32 128-bit registers

Table 2. Composition of RISC-V Extension ISA

Name	Numbers	Description
M	8	Contains 4 multiplication, 2 division, and 2 remainder operation instructions
A	11	Contains atomic operation instructions, such as read-modify-write, compare-exchange, etc
F	26	Contains single precision floating point instructions
D	26	Contains double precision floating point instructions
Q	26	Contains Quad Precision floating-point instructions
C	46	Compressed instruction set, in which the instruction length is 16 bits, the main purpose is to reduce the code size

X25519 is one of ECDH key negotiation protocols based on Curve255119 elliptic curve. Compared with the traditional ECDH key negotiation protocol, the most remarkable feature of X25519 protocol is that it can calculate the x coordinate of kP only by relying on the x coordinate of point P on the elliptic curve. The idea of constructing ECDH key negotiation protocol using only x-coordinate came from the foundational paper published by Victor Miller in 1985 [15].

Table 3. Operation Instructions

Instructions	Meanings
$ADD(a, b, c, s)$	$s = a + b$, and overflow value saved to c
$ADDC(a, b, c, s)$	$s = a + b + c$, and overflow value saved to c
$SUB(a, b, c, s)$	$s = a - b$, and underflow value saved to c
$SUBC(a, b, c, s)$	$s = a - b - c$, and overflow value saved to c
$MUL(a, b, h, l)$	$h = (a \times b)_{hi}$, $l = (a \times b)_{lo}$

2.3 RISC-V and VisionFive

RISC-V instruction set architecture (ISA) was proposed by Andrew Waterman and others at the University of California, Berkeley in 2010. At present, its application has covered many fields such as IoT devices, desktop computers, high-performance computers, and so on. RISC-V is an open-source instruction set architecture, and its goal is to become Linux in the field of instruction set architecture. Andrew Waterman mentioned in his doctoral dissertation [16] that the existing instruction sets are all commercial instruction sets protected by patents and are not open, which not only restricts competition, but also curbs innovation, and brings certain obstacles to the development of the chip industry; On the other hand, the existing instruction sets have a long history and have undergone several twists and turns in the development process. Therefore, most of them carry a lot of historical burdens, resulting in the instruction sets being quite complex and not suitable for academic research. In this case, an open new instruction set architecture can not only be freely used by more people but also avoid many historical problems.

At the beginning of design, RISC-V instruction set is considered to be applicable to education and scientific research as well as to meet the application requirements in industrial scenarios. In order to achieve the above two purposes, RISC-V instruction set adopts the form of combining a basic instruction set and extension instruction sets, taking a basic instruction set as the core, and then adding different extension instruction sets according to functional requirements, just like adding plug-ins to an application program, which fully embodies the idea of modularization. RISC-V ISA has four basic instruction sets, as shown in Table 1, and six standard extension instruction sets, as shown in Table 2.

VisionFive is a Linux single-board computer based on RISC-V ISA launched by the technology company StarFive. It can be widely used in edge computing, intelligent home appliances, intelligent monitoring, industrial robots, traffic management, intelligent logistics, wearable devices, network communications, and other fields. Visionfive embedded development board is equipped with 64-bit JH7100 CPU and 8GB LPDDR4 memory. The JH7100 processor is equipped with SiFive U74 dual-core and fully implements the RV64GC instruction set specification (where G is the initial letter of general and represents the specific

combination of "IMAFD"). The operating frequency reaches 1GHz, which can provide strong computing capability for embedded intelligent devices such as edge computing, deep learning, and Internet of Things.

3 Methodology

This section describes our implementation method of the X25519 protocol based on RISC-V in detail. We first introduce the representation of large integers, including the concept and principle of radius limb. After that, we demonstrate how to calculate large integers at the finite field level under the representation method of radius limb, including addition, subtraction, multiplication, square, inverse, modular reduction, etc. Finally, we carry out scalar multiplication of Curve25519 elliptic curve in single coordinate by Montgomery Ladder algorithm to fully implement X25519 key agreement protocol.

3.1 Radix-2^{64} Limb Representation of Large Integer

At present, a large number of works have done relevant research on the representation scheme of large integers. Fundamentally, the large number representation scheme needs to fully consider the characteristics of the target platform, so as to select the appropriate digital representation scheme. For the 255-bit prime field elements used by curve25519, [2] initially proposed a scheme of Radix-$2^{25.5}$ to represent the field elements in the 32-bit architecture, which is widely spread and widely used in the industry. For the Radix-$2^{25.5}$ scheme, each 255-bit integer is composed of 10 limbs, including 5 length bits of 25 bits and the other 5 length bits of 26 bits. More formally, the field element f is represented by the following equation:

$$f = f_0 + 2^{26} f_1 + 2^{51} f_2 + 2^{77} f_3 + 2^{102} f_4 + 2^{128} f_5 + 2^{153} f_6 + 2^{179} f_7 + 2^{204} f_8 + 2^{230} f_9, \tag{5}$$

where $0 \leq f_{2i} < 2^{26}$ and $0 \leq f_{2i+1} < 2^{25}$ for $0 \leq i \leq 4$. For the calculation of a large integer represented by multiple limbs, if the lower limb exceeds the representable range, it is necessary to carry to the higher limb. This carry is generally highly continuous (called carry propagation), which is not conducive to the operation efficiency of modern CPU.

The Radix-$2^{25.5}$ representation scheme has significant advantages, because, for a 26-bit number a, $19a$ is still within the range of 32-bit integers, so that the carry propagation can be effectively delayed until the end of the whole multiplication, thereby improving the efficiency of the algorithm to a certain extent. However, considering our RV64GC instruction set architecture and the processing capability of SiFive U74 core, this representation is not necessarily the best

choice. Therefore, we chose Radix-2^{64} to represent the domain elements. For any f belonging to $\mathbb{F}_p$, it is composed of four 64-bit limbs. The equation is as follows:

$$f = f_0 + 2^{64} f_1 + 2^{128} f_2 + 2^{192} f_3, \tag{6}$$

where $0 \leq f_i < 2^{64}$ for $0 \leq i \leq 3$.

3.2 Implementation of Finite Field Arithmetic

As we all know, the underlying finite field arithmetic is the core that affects the efficiency of the entire elliptic curve cryptosystem. Therefore, optimizing the performance of X25519 key agreement algorithm is to optimize the underlying finite field arithmetic. In this paper, finite field addition, finite field subtraction, and finite field multiplication are implemented based on Radix-2^{64}, and fast modulo reduction in finite field is implemented according to the characteristics of modulo $p = 2^{255} - 19$.

In order to adapt to the Radix-2^{64} scheme used in this paper and improve the efficiency of the code, we have split up the calculation of finite field arithmetic and written the required calculation operations in the form of instructions in C language macro statements. The operation instructions involved in the experiment are shown in the Table 3.

Finite Field Addition. For the implementation of finite field arithmetic operation on the bottom of an elliptic curve, it is usually modified according to the specific large integer representation scheme to achieve a certain degree of optimization effect. For example, work [17] uses 8 32-bit fixed points to represent a large integer of 255 bits, thus redundant data of 1 bit. With the redundant bits of 1-bit, the input and output of the algorithm can be controlled at 256 bits. In the specific calculation process, only the large integer needs to be reduced to 256 bits, and only 256 bits can be reduced to $\mathbb{F}_p$, where $p = 2^{255} - 19$, when calculating the final scalar multiplication result. This technique can be used to cleverly simplify the rounds of reduction of large integers, thereby increasing the overall computational efficiency. However, this method requires an additional register to hold the overflow of 256-bit large integer addition results, which to some extent limits the overall performance. This paper uses 4 64-bit fixed-point numbers to represent a 255-bit large integer. Similar to the 8 32-bit fixed-point representation schemes, our representation scheme also has 1-bit redundancy, so we first tried the redundancy representation scheme in [17]. However, after many experiments, the performance of this scheme is not optimal on our RISC-V platform, so we simply chose the best implementation scheme.

Two 255-bit numbers are added in our finite-field addition algorithm, which means that the highest bit of the large integer addition input must be guaranteed to be zero to compute a 256-bit result. We don't need to immediately reduce the 256-bit calculation to 255-bit, but only when it is necessary to affect the correctness of the algorithm, in order to improve the overall efficiency. The flow

chart for finite field addition is shown in the Fig. 4, where the large integer addition $C = A + B$ uses one $ADD()$ macro instruction, two $ADDC()$ macro instructions and two additional simple additions.

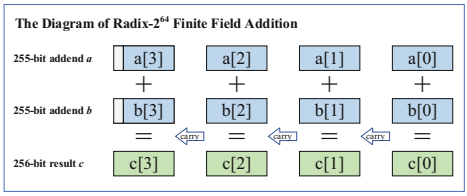

Fig. 4. Finite Field Addition

In addition, since RISC-V architecture does not provide the same functionality as CF in Program Status Word in X86 CPUs, programmers need to use software methods to determine whether data is overflowing or not. A common way to determine overflow is to compare the size of the operands and results. For example, as shown in Algorithm 1, in unsigned 64-bit integer addition $c = a + b$, if c is smaller than a or b instead, we can infer that a data overflow must have occurred. We can use a variable carry to hold the carry value, and when the carry occurs, it is set to 1, representing the 65th position of c. This makes it a carry operation in addition.

Algorithm 1. The carry generation method in addition

Input:
Two addends a, and b;
Output:
The result c, and the carry cy generated by addition;
1: $c = a + b$
2: **if** $c < a$ **then**
3: $cy = 1$
4: **else**
5: $cy = 0$
6: **end if**

Finite Field Subtraction. Similar to finite field addition, to avoid introducing additional registers to handle the "debits" that result from the calculation, the input for finite field subtraction is limited to 255-bit so that the overflow from the subtraction process can be recorded using the redundant space of up to 1 bit. In large integer subtraction $C = A - B$, if A is less than B, the highest bit of the result will also be set to "1", indicating an underflow. However, unlike the addition that does not require immediate reduction of 256-bit to 255-bit,

the underflow handling and reduction of finite-field subtraction is not common to addition, multiplication, etc. Therefore, we have separately integrated the steps of reduction in the calculation of finite-field subtraction to make the result $C \in \mathbb{F}_p$, that is, to output the result of a 255-bit calculation. The flow chart for finite-field subtraction is shown in the Fig. 5. First, use the conventional method to calculate the 255-bit large integer subtraction $C = A - B$, and get the 256-bit calculation result. Then, a *while*() loop is used to eliminate the possible "1" generated by the highest bit, that is, simply set it to zero, and then subtract 19 from the remaining large integer (because $2^{256} \equiv 19 \pmod{2^{255} - 19}$). Since the remaining large integers may be less than 19, the *while*() loop will execute a maximum of 2 times.

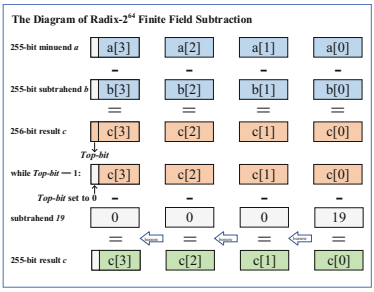

Fig. 5. Finite Field Subtraction

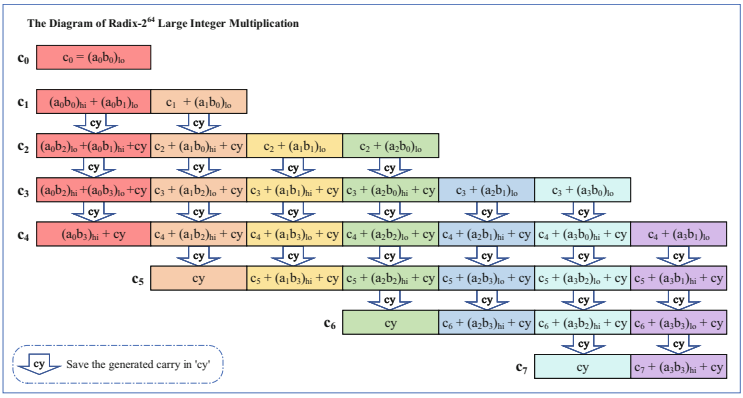

Fig. 6. Large Integer Multiplication

Finite Field Multiplication and Square. Finite field multiplication is the most computationally intensive and time-consuming underlying operation

besides inversion, which has a significant impact on the performance of any elliptic curve cryptography system. Therefore, optimizing the calculation of multiplication is essential.

For the elliptic curve Curve25519, it is applied to the finite field $\mathbb{F}_p$, where $p = 2^{255} - 19$. As explained above, we use the Radix-2^{64} scheme, where a field element consists of four 64-bit numbers, to represent a large integer A, that is $A \in [0, 2^{256})$. Unlike finite field addition and subtraction, restricting the input of a finite field multiplication to 255-bit does not effectively reduce the use of additional variables, but rather results in some loss of efficiency due to additional reduction of the input. Therefore, in our multiplication calculation, the input can be 256-bit. For large integer multiplication $C = A \times B$ (where $A, B \in [0, 2^{256})$), there is $C \in [0, 2^{512})$.

The basic unit that we use in the implementation of finite-field multiplication is the $MUL(a, b, h, l)$ macro instruction defined in Table 3. The specific implementation process is shown in the Fig. 6. As shown in the figure, we perform iteration calculations from left to right on a macroscopic level and update the calculation results. In other words, the rightmost grid of each line is the final calculation result of each limb. In addition, each column is calculated from top to bottom. In a nutshell, it is calculated in the order of rainbow colors. First, calculate the red column, then calculate the orange column, and so on, and finally iterate to calculate the final large integer multiplication result.

For the calculation of the finite field square, since the two multiplier inputs are the same, that is, for the large integer multiplier $C = A \times B$ (where $A = B$), there is $A_i \times B_j = A_j \times B_i$. Therefore, there is a calculation method that can reduce the overall calculation steps by reusing intermediate results. To improve the overall performance of the algorithm, much literature [17,18] has implemented the finite field square operation separately. The process of finite field square is divided into the following three steps:

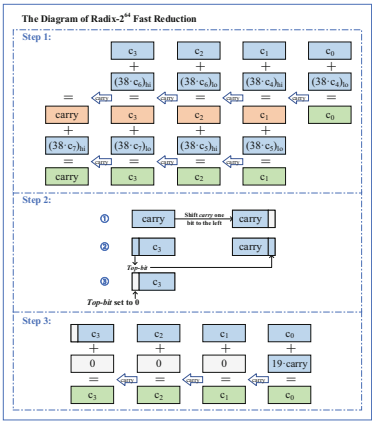

Fig. 7. Fast Reduction

- Firstly, when $i < j$, $A_i \times B_j$ (where $A = B$) is calculated, which is similar to the large integer multiplication mentioned above;
- Secondly, shift the above calculation result to the left by one bit. At this time, we have obtained the calculation result of $A_i \times B_j + A_j \times B_i$ $(i \neq j)$ (that is, twice as much as $A_i \times B_j$);
- Finally, for each i, compute $A_i \times A_i$, and add the results to the above results.

In the above calculation, the performance bottleneck lies in the left shift operation of the second step. Since the large integer representation scheme we use is Radix-2^{64} with 64-bit limb size, no redundant bits are reserved for each limb to use. Therefore, the left shift operation needs to receive the highest bit data from the lower level branch and send its highest bit data to the lowest level of the higher level branch, which has a huge performance impact. Our experimental results show that although the algorithm implemented for finite field squares alone can effectively reduce the number of simple multiplications, its overall performance is slightly weaker than that of using the multiplication algorithm directly for square calculations due to the influence of shift operation. In our experiment, calculating squares directly using the multiplication algorithm improves the performance of the algorithm by about 4% compared to implementing the algorithm for squares alone.

Fast Reduction. As we have learned from the previous section, the input of a large integer multiplication is two large 256-bit integers, which produce a 512-bit product output. For large integer multiplication $C = A \times B$ (where $A, B \in [0, 2^{256})$), there is $C \in [0, 2^{512})$. Further, C can be expressed as:

$$\begin{aligned} C &\equiv \sum_{i=0}^{7} C_i \times 2^{64i} \pmod{p} \\ &\equiv \sum_{i=4}^{7} C_i \times 2^{64i} + \sum_{i=0}^{3} C_i \times 2^{64i} \pmod{p} \end{aligned} \tag{7}$$

From $p = 2^{255} - 19$, we can get: $2^{255} \equiv 19 \pmod{p}$ and $2^{256} \equiv 38 \pmod{p}$ Then, Eq. 7 can be further reduced to:

$$\begin{aligned} C &\equiv \sum_{i=0}^{3} 38 \times C_{i+4} \times 2^{64i} + \sum_{i=0}^{3} C_i \times 2^{64i} \pmod{p} \\ &\equiv \sum_{i=0}^{3} (C_i + 38 \times C_{i+4}) \times 2^{64i} \pmod{p} \end{aligned} \tag{8}$$

It can be seen from Eq. 8 that the 512-bit large integer multiplication result C is composed of 8 64-bit fixed-point numbers. For the large integer multiplication result C, we can take the digit with the higher 256 bits $(c_4 c_5 c_6 c_7)$, multiply it

by 38, and add it to the lower 256 bits of C ($c_0c_1c_2c_3$). In this way, we get new calculation results, and it is obvious that the maximum length will not exceed 262 bits, of which the highest 6 bits are stored in a separate 32-bit variable, *carry*. At this time, we have made the first round of reduction for C, which is reduced from 512 bits to no more than 262 bits. Next, consider how to reduce 262 bits to 256 bits. If we continue to multiply the carry by 38 and add it to the lower 256 bits, a new 1-bit overflow will occur, which will trigger a new round of reduction. Therefore, we adopted an ingenious method, that is, we shifted the entire *carry* to the left by one bit, and then moved the highest bit of the lower 256 bits part to the lowest bit of the *carry*. In this way, the lower 256 bits have 256 positions, but only 255 bits of information are saved. The value of carry cannot exceed 7 bits at most. According to equation $2^{255} \equiv 19 \pmod{p}$, carry can be multiplied by 19 and added to the lower 255 bits to obtain a calculation result not exceeding 256 bits. The overall fast reduction process is shown in Fig. 7.

3.3 Implementation of Scalar Multiplication Arithmetic

The most traditional scalar multiplication scheme for variable base points of elliptic curves is the Double-add scheme, which receives cache-timing side channel attacks easily and is inefficient due to the input-related computation delay. The Montgomery Ladder scheme can well resist side-channel attacks due to the fixed latency of the computation. This paper uses Montgomery Ladder algorithm to realize scalar multiplication of Curve25519 elliptic curve. The flow of the algorithm is shown in Algorithm 2, where cswap() is a conditional exchange function and *swap* is 1 to exchange the values of two variables. This function is completed using the scheme of literature [10], which also guarantees fixed calculation delay. In the Montgomery Ladder algorithm, there is a fixed data variable a_{24}, whose specific value in Curve25519 elliptic curve is:

$$a_{24} = \frac{486662 - 2}{4} = 121665. \tag{9}$$

Since a large integer multiplication in the Curve25519 Montgomery Ladder algorithm process requires the fixed value $a_{24} + 1 = 121666$, which does not exceed the maximum length of 17 bits, the full $256bits \times 256bits$ multiplication is not required. In *step 22* of Algorithm 2, we separately implemented a simplified version of the modular multi-add algorithm for calculating the large integer multiplication and addition algorithm $C(256bits) = A(256bits) \times B(64bits) + D(256bits)$, which omits the corresponding part of $A \times (b_1, b_2, b_3)$ calculation, and improves the overall efficiency of the algorithm. In addition, in order to improve the utilization of registers, reduce the swapping in and swapping out of memory, so as to make full use of the CPU pipeline and improve the computing efficiency, we fine tuned the order of some computing steps in the Montgomery Ladder algorithm, so that it can adapt to our RISC-V computing platform.

Algorithm 2. The X25519 key negotiation function based on Montgomery Ladder algorithm

Input:
The 255-bit length scalar k, and x coordinate of random point P: u;
Output:
The x coordinate of scalar multiplication $k \cdot P$;

$x_1 = u, x_2 = 1, x_3 = u$
$z_2 = 0, z_3 = 1, swap = 1$
for $i = 254$ *to* 0 **do**
  $k_i = (k >> 1)\&1$
  $swap = swap \oplus k_i$
  $(x_2, x_3) = \text{cswap}(swap, x_2, x_3)$
  $(z_2, z_3) = \text{cswap}(swap, z_2, z_3)$
  $swap = k_i$
  $tmp1 = x_2 - z_2$
  $x_2 = x_2 + z_2$
  $tmp0 = x_3 - z_3$
  $z_2 = x_3 + z_3$
  $z_3 = tmp0 \times x_2$
  $z_2 = tmp1 \times z_2$
  $tmp0 = tmp1^2$
  $tmp1 = {x_2}^2$
  $x_3 = z_3 + z_2$
  $z_2 = z_3 - z_2$
  $x_2 = tmp1 \times tmp0$
  $tmp1 = tmp1 - tmp0$
  $z_2 = {z_2}^2$
  $tmp0 = tmp1 \times 121666 + tmp0$
  $x_2 = {x_2}^2$
  $z_3 = x_1 \times z_2$
  $z_2 = tmp1 \times tmp0$
end for
$(x_2, x_3) = \text{cswap}(swap, x_2, x_3)$
$(z_2, z_3) = \text{cswap}(swap, z_2, z_3)$
return $x_2 \times ({z_2}^{-1})$

4 Performance Evaluation

In this section, the experimental results of the optimized X25519 function on the VisionFive RISC-V development board are presented and compared with other work. The necessary introduction to the VisionFive development board has been made earlier. Table 4 gives our hardware and software experimental environment, including system version, toolchain, hardware configuration parameters, etc.

Table 4. The Platform Configuration of V-Curve25519

Type	Model
OS	Linux Fedora release 33 (Rawhide)
Tool Chain	gcc (Red Hat 10.3.1-1)
CPU	SiFive U74 2-core 64-bit RV64GC @ 1.0GHz
Memory	8 GB LPDDR4
Power	About 15W

Table 5. Experiment Results

Operation Type	Cycles
Finite Field Addition	7
Finite Field Subtraction	10
Finite Field Multiplication	86
Finite Field Square	97
Finite Field Inversion	22304
Finite Field Mul121666AddNum	25
Curve25519 Scalar Multiplication	295967

Table 6. Scalar Multiplication Performance Comparison

	Platform	Curve	Cycles	TP (ops/s)	LAT (ms)
Liu et al. [19]	Cortex-M4F @ 400 MHz	Four$\mathbb{Q}$	880,642	350	2.79
Wei et al. [20]	Cortex-M0 @ 48 MHz	Four$\mathbb{Q}$	1,972,000	24	41.08
Nishinaga et al. [21]	Cortex-M0 @ 48 MHz	Curve25519	5,164,352	9	107.64
Nishinaga et al. [21]	Cortex-M0+ @ 72 MHz	Curve25519	4,209,866	17	58.48
Hayato et al. [22]	Cortex-M4 @ 48 MHz	Curve25519	907,240	52	18.90
Hayato et al. [22]	Cortex-M4 @ 72 MHz	Curve25519	1,003,707	71	13.94
Hayato et al. [22]	Cortex-M4 @ 84 MHz	Curve25519	894,391	93	10.65
Stefan [23]	Hifive1 @ 320MHz	Curve25519	5,389,988	59	16.84
OpenSSL [11]	VisionFive @ 1GHz	Curve25519	402,000	2,490	0.40
Ours	VisionFive @ 1GHz	Curve25519	**295,967**	**3,378**	**0.29**

4.1 Experiment Results

For the 64-bit RISC-V instruction set architecture, we completed the performance optimization of the key agreement algorithm based on Curve25519 curve from the bottom finite field to the top scalar multiplication. The scheme in this paper can be used for the identity authentication and security computing of the Internet of Vehicles, making up for the shortcomings of existing algorithms in the adaptation of RISC-V 64-bit platform.

As shown in Table 5, we show the number of *Cycles* consumed in each operation of our implementation scheme. It can be seen from the table that the finite field subtraction operation consumes 3 more instruction cycles than the addition operation. This is because the reduction of subtraction is different from addition and multiplication and needs to be implemented separately. In addition, the square of a finite field is implemented by simply passing two identical parameters to the multiplication of a finite field, and the compiler completes the corresponding optimization, which saves 11 instruction cycles compared with the multiplication of a finite field. The inversion of finite fields and the Montgomery ladder algorithm are based on the addition, subtraction, multiplication, and square of finite fields. Therefore, the focus of subsequent research is still to continue to explore the optimal implementation of finite field multiplication.

4.2 Performance Comparison

Table 6 summarizes the X25519 unknown point scalar multiplication we implemented based on the VisionFive development board for Diffie-Hellman key negotiation protocol, and compares it with other platforms. This paper mainly evaluates the performance of our scheme from the following three aspects:

- Cycles: It represents the number of instruction cycles required to calculate X25519 function once;
- Throughput (ops/s): It indicates the number of X25519 key negotiation functions completed by the RSIC-V 64-bit CPU in a unit time;
- Latency (ms): It refers to the time required for X25519 on the RSIC-V 64-bit CPU platform from the calculation request to the end of the calculation.

As can be seen from Table 6, due to the limited performance of computing platforms and the algorithm implementation itself, the number of instruction cycles of elliptic curve scalar multiplication on most embedded platforms ranges from hundreds of thousands to millions, and the throughput is about tens of times per second. Even the fastest elliptic curve Four$\mathbb{Q}$ is claimed to have a computing speed of only about 350 times per second on ARM Cortex-M4F platform, which shows that the computing performance of related cryptographic algorithms on specific resource-constrained platforms is still insufficient to support massive computing requests. After our test on the VisionFive development board, the overall computing performance of the X25519 open-source crypto library OpenSSL [11] shows that it is the implementation of the most advanced X25519 key agreement algorithm on the RISC-V 64-bit platform, with the minimum number of single computing instruction cycles and delay, and the maximum computing throughput.

OpenSSL [11] is currently the most popular and widely used tool for SSL cryptographic libraries, and it is also the state-of-the-art implementation of cryptography algorithm on our test platform. It provides a generic, robust, and fully functional tool suite to support the implementation of the SSL/TLS protocol. The X25519 key negotiation algorithm is also provided in OpenSSL

and the performance of scalar multiplication can be tested using the command "*openssl speed ecdhx*25519".

In Table 6, the experimental results of the scheme and the implementation of OpenSSL on our experimental platform, VisionFive, are detailed. From the table, we can see that the performance of OpenSSL can be calculated approximately 2490 times per second by X25519 scalar multiplication. Compared with OpenSSL, our experimental results achieved higher throughput, reaching 3378 times per second, bringing about 35% performance improvement. Furthermore, our computing latency is smaller than that of OpenSSL implementation, and it can withstand the risk of side-channel attacks to some extent.

5 Conclusion

Since the introduction of the RISC-V instruction set architecture with open attributes, it has become a hot breeze in the IoT world. In this paper, based on RISC-V 64-bit processor, the X25519 key negotiation function is fully optimized, and the throughput reaches 1.35 times that of OpenSSL cryptographic library standard implementation, which has obvious performance advantages. On the other hand, it also takes into account a certain anti-side channel attack capability. In the future, we will focus on the computational performance of PQC on RISC-V architecture and continue to study the efficient implementation of related algorithms.

References

1. Gates, B., Myhrvold, N., Rinearson, P., Domonkos, D.: The road ahead (1995)
2. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976)
3. Doyle, B., Bell, S., Smeaton, A.F., McCusker, K., O'Connor, N.E.: Security considerations and key negotiation techniques for power constrained sensor networks. Comput. J. **49**(4), 443–453 (2006)
4. Kerry, C.F., Gallagher, P.D.: Digital signature standard (DSS). FIPS PUB 186-4 (2013)
5. Goldreich, O.: Secure multi-party computation. Manuscript. Preliminary version, vol. 78, p. 110 (1998)
6. Chandra, S., Paira, S., Alam, S.S., Sanyal, G.: A comparative survey of symmetric and asymmetric key cryptography. In: 2014 International Conference on Electronics, Communication and Computational Engineering (ICECCE), pp. 83–93. IEEE (2014)
7. Suga, Y.: SSL/TLS status survey in japan-transitioning against the renegotiation vulnerability and short RSA key length problem. In: 2012 Seventh Asia Joint Conference on Information Security, pp. 17–24. IEEE (2012)
8. Koblitz, N.: Elliptic curve cryptosystems. Math. Comput. **48**(177), 203–209 (1987)
9. Bernstein, D.J.: Curve25519: new Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Heidelberg (2006). https://doi.org/10.1007/11745853_14

10. Langley, A., Hamburg, M.: Elliptic curves for security, order, vol. 500, p. 39081 (2016)
11. OpenSSL Software Foundation: OpenSSL Cryptography and SSL/TLS Toolkit (2016). http://www.openssl.org/
12. Rescorla, E.: The transport layer security (TLS) protocol version 1.3. Technical report (2018)
13. RISC-V International®. RISC-V international (2022). https://riscv.org/
14. Diffie, W., Hellman, M.E.: Multiuser cryptographic techniques. In: Proceedings of the 7–10 June 1976, National Computer Conference and Exposition, pp. 109–112 (1976)
15. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39799-X_31
16. Waterman, A.S.: Design of the RISC-V instruction set architecture. University of California, Berkeley (2016)
17. Dong, J., Zheng, F., Cheng, J., Lin, J., Pan, W., Wang, Z.: Towards high-performance X25519/448 key agreement in general purpose GPUs. In: 2018 IEEE Conference on Communications and Network Security (CNS), pp. 1–9. IEEE (2018)
18. Düll, M., et al.: High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. Des. Codes Crypt. **77**(2–3), 493–514 (2015)
19. Liu, Z., Longa, P., Pereira, G.C., Reparaz, O., Seo, H.: Four$\mathbb{Q}$ on embedded devices with strong countermeasures against side-channel attacks. IEEE Trans. Dependable Secure Comput. **17**(3), 536–549 (2018)
20. Zhang, W., Lin, D., Zhang, H., Zhou, X., Gao, Y.: A lightweight FourQ primitive on ARM cortex-M0. In: 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), pp. 699–704. IEEE (2018)
21. Nishinaga, T., Mambo, M.: Implementation of μNACL on 32-bit ARM cortex-M0. IEICE Trans. Inf. Syst. **99**(8), 2056–2060 (2016)
22. Fujii, H., Aranha, D.F.: Curve25519 for the cortex-m4 and beyond. In: Lange, T., Dunkelman, O. (eds.) LATINCRYPT 2017. LNCS, vol. 11368, pp. 109–127. Springer, Cham (2017). https://doi.org/10.1007/978-3-030-25283-0_6
23. van den Berg, S.: RISC-V implementation of the NACL-library. Ph.D. dissertation, Master Thesis, vol. 1, no. 1 (2020)