DPF-ECC: A Framework for Efficient ECC With Double Precision Floating-Point Computing Power

Lili Gao, Fangyu Zheng, Rong Wei, Jiankuo Dong, Niall Emmart, Yuan Ma, Jingqiang Lin, *Senior Member, IEEE*, and Charles Weems, *Senior Member, IEEE*

***Abstract*— Used ubiquitously in a huge amount of security protocols or applications, elliptic curve cryptography (ECC) is one of the most important cryptographic primitives, featuring efficiency and short key size compared with other public-key cryptosystems such as DSA and RSA. However, as a computation-intensive public-key cryptographic primitive, ECC arithmetic is still the bottleneck that restrains the overall performance of the end applications. In this paper, instead of the conventional and straightforward integer-based methods, we present a general framework to accelerate ECC schemes over prime field, called DPF-ECC, that deeply exploits double precision floating-point (DPF) computing power. The DPF-ECC framework finely manages each bit of the DPF numbers and minimizes the overhead brought by additional data format conversion, by making use of the DPF representation, the rounding operations, and fused multiply-add instruction supported by the IEEE 754 floating point standard. We also conduct two comprehensive case studies on Crandall primes and Solinas primes to demonstrate how the DPF-ECC framework is applied to the prevailing ECC schemes. To evaluate the proposed DPF-ECC framework in the real world, leveraging the floating-point computing power of GPUs, we implement Curve25519/448 and Edwards25519/448, the popular ECC schemes widely used in TLS 1.3, SSH, etc. The experimental result in Tesla P100 achieves a record-setting performance that outperforms the existing fastest integer work with 2x to 3x throughput. With dependency only on the very commonly supported IEEE 754 floating point standard, DPF-ECC framework can be a very competent and promising candidate for ECC implementation in most of general-purpose platforms.**

***Index Terms*— Elliptic curve cryptography, floating-point arithmetic, graphics processing unit.**

Manuscript received March 2, 2021; revised May 27, 2021; accepted June 30, 2021. Date of publication July 21, 2021; date of current version August 17, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61902392, in part by the National Science Foundation (NSF) under Grant CCF-1525754, and in part by the National Key Research and Development Program of China under Grant 2017YFB0802100. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Debdeep Mukhopadhyay. *(Corresponding author: Fangyu Zheng.)*

Lili Gao, Fangyu Zheng, Rong Wei, and Yuan Ma are with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China, also with the School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China, and also with the Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing 100093, China (e-mail: zhengfangyu@iie.ac.cn).

Jiankuo Dong is with the School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu 210049, China.

Niall Emmart and Charles Weems are with the College of Information and Computer Sciences, University of Massachusetts Amherst, Amherst, MA 01003, USA.

Jingqiang Lin is with the School of Cyber Security, University of Science and Technology of China, Hefei, Anhui 230026, China.

Digital Object Identifier 10.1109/TIFS.2021.3098987

I. Introduction

THE proliferation of the Internet has given rise to software distribution, e-commerce, and other industries that serve huge-scale users. Used ubiquitously in security-sensitive applications of these industries, public-key cryptosystems have been indispensable in cybersecurity. Elliptic Curve Cryptography (ECC) [1], [2], widely used in key management and digital signatures for privacy protection and secure communications over the Internet, has gradually become the cornerstone of the public-key cryptosystems. ECC provides more computationally efficiency and greater security per bit increase in key size than RSA. In recent years, a growing number of security protocols have introduced cryptographic primitives based on elliptic curve groups. At the RSA Conference 2005, the National Security Agency (NSA) announced Suite B which uses ECC for digital signature and key agreement. The NIST standardized the Elliptic Curve Digital Signature Algorithm (ECDSA) in [3] for verifying the authenticity of digital messages or documents. In 2016 and 2017, RFC7748 [4] and RFC8032 [5] released by the Internet Engineering Task Force (IETF) recommended Curve25519/448 and Edwards25519/448 for an Elliptic Curve Diffie-Hellman (ECDH) protocol and the Edwards-curve Digital Signature Algorithm (EdDSA). Curve25519/448 and Edwards25519/448 are the most prevalent ECC schemes and are widely used in many protocols such as TLS 1.3 [6] and SSH [7], [8]. ECC-based schemes have been applied ubiquitously in numerous Internet security protocols and applications, which are of great importance.

A. Motivations

1) Urgent Demands for Cryptographic Services: In fact, ECC algorithms whose chief cryptographic primitive is point multiplication (PM), are far more computationally expensive than symmetric algorithms, e.g., block ciphers and hash functions. Some of the protocols even require multiple operations of the ECC primitives, for example, J-PAKE [9] requires 11 point multiplications in a single session.

The ECC algorithm is not only computationally intensive but also urgently required. For example, Alipay set a record by processing up to 583,000 payment transactions per second on "Double-Eleven", the Chinese online "Black-Friday" in 2020 [10]. In each payment transaction, the sensitive information (e.g., buyer privacy) shall be encrypted by a key which is agreed between Alipay and buyer, the buyer shall be authenticated and the deal shall be notarized via

digital signatures. Both key agreement and digital signature are heavily dependent on the "expensive" ECC primitives. This is a daunting task.

2) Rapid Evolution of Parallel Platforms: Single instruction, multiple threads (SIMT) introduced by NVIDIA in 2006 is an execution model that is used in graphics processing units (GPUs) for parallel computing. SIMT utilizes thread-level parallelism, i.e. multiple independent threads execute concurrently using a single instruction. Both high-definition 3D graphics processing and deep learning applications require high-speed floating-point processing capabilities. From 2010 to the present, the floating-point computing power of NVIDIA GPUs has grown over ten-fold, from 1,345 (Fermi architecture) Giga Floating-point Operations Per Second (GFLOPS) to 15,000 (Volta architecture) GFLOPS.

B. Technical Challenges and Contributions

Based on this fact, researchers are committed to studying high-performance cryptographic computing technologies on GPUs, including ECC [11]–[15], RSA [11], [16]–[19], post-quantum cryptography [20], [21] and even homomorphic encryption [22]–[24] and demonstrated that GPUs can outperform the conventional cryptographic implementations based on CPU or even hardware. The industries also pay increasingly attention to GPU-accelerated hardware security modules that provide cryptographic services. For example, HSM-ZJ2014 [25], a GPU-accelerated hardware security module has been validated against FIPS 140-2 for commercial usages.

Noting that parallel platforms (e.g., GPUs) offer stronger and more promising floating-point computing power than integer ones, several researchers attempted to accelerate public-key cryptographic algorithms with floating-point computing power. However, compared to integer-based works, the less regular digital representation of floating-point numbers causes difficulty on format conversion, data merging, and big-number reduction in the process of calculation.

Targeting the RSA algorithm, Zheng *et al.* [16] utilized floating-point arithmetic to accelerate modular exponentiation, which demonstrated floating-point-based scheme can outperform the conventional integer-based ones. After this work, three follow-up papers [17]–[19] made further contributions to the floating-point-based RSA and achieved the current best performance.

Compared with the RSA algorithm whose main overhead is only the modular exponentiation, ECC schemes require more careful consideration and more fine grained control, because of the relatively complex curve-level arithmetic and less regular underlying finite field arithmetic, especially with the floating-point computing power. As a result, there has been limited research into floating-point based ECC implementations. However, as floating-point processing power evolves, fully utilizing the floating-point processing resources holds great promise for the ECC implementation. Bernstein *et al.* are the first to utilize the floating-point processing power of GPUs for 280-bit ECM (Elliptic Curve Method) [12]. They performed one modular multiplication by a group of 28 threads and each thread processes a 10-bit limb. However, the performance is only 1/6 of their follow-up integer-based work [26].

In this paper, we propose a novel framework that applies double precision floating-point arithmetic to ECC schemes and present comprehensive case studies. The main contributions of this work are three-fold:

- We design a DPF-ECC framework, a general scheme utilizing the double precision float computing power to accelerate modular arithmetic of ECCs.
- We comprehensively conduct case studies on how the DPF-ECC framework is applied to the prevailing ECC schemes whose p is a Crandall prime ($p = 2^k - \sigma$) or a Solinas prime and give design principles and best parameter settings.
- Utilizing multiple optimization techniques, we have a complete implementation of unknown point multiplication of Curve25519/448, known point and unknown point multiplication of Edwards25519/448 on GPU computing platforms. Our experiment sets a record for both throughput and latency, achieves over 19 times speedup (considering the platform differences) of [12] in which Bernstein *et al.* conducted the implementation of ECC with floating-point computing power on GPU. The experimental results also outperform the existing fastest integer-based approaches with 2x to 3x throughput.

Our work shows that the floating-point computing power of parallel platforms can be an excellent candidate for public-key cryptography acceleration. In fact, our approach can be easily extended for ECC schemes to many computing platforms, since we rely mainly on the standard floating-point representation and arithmetic that most general-purpose platforms support.

The rest of the paper is organized as follows. Section II covers basic knowledge. Section III presents the DPF-ECC framework. Section IV focuses on the case study. Section V gives implementation details and performance evaluation and comparison. Section VI concludes the paper.

II. Background

In this section, we describe the background required for this paper.

A. Elliptic Curves

Based on the elliptic curve discrete logarithm problem (ECDLP), i.e., finding the discrete logarithm (d) of a random elliptic curve element ($Q = [d]P$) with respect to a publicly known base point P is infeasible, ECC is a public-key technology that offers performance advantages at higher security levels. And for most of the ECC schemes, the core operation is *point multiplication*, i.e., computing a multiple of a point $Q = [d]P$. According to different security considerations, the point P may be known or unknown.

1) Weierstrass Curves: The Weierstrass curve

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

is a very standard form that is introduced in almost every textbook and adopted in many ECC standards, e.g., Elliptic Curve Diffie Hellman (ECDH) in NIST Special Publication 800-56A [27] and Elliptic Curve Digital Signature Algorithm (ECDSA) [3]. But in recent years, two forms of elliptic curves are being favored by the cryptographers and industry due to their performance and "security", Montgomery curve

and twisted Edwards curve. Next, we give a brief introduction to these two forms of elliptic curve.

2) Montgomery Curves: Let $\mathbb{F}_p$ be a prime field of odd characteristics; a Montgomery curve [28] over $\mathbb{F}_p$ is defined as

$$E_{a,b}/\mathbb{F}_p : by^2 = x^3 + ax^2 + x. \tag{2}$$

where $a, b \in \mathbb{F}_p$, $a \neq \pm 2$.

Taking a scalar d and an x-coordinate of point P as input and producing an x-coordinate of $[d]P$ as output, a Montgomery ladder is an efficient x-coordinate-only algorithm to calculate point multiplications. In combination with Diffie-Hellman key exchange protocol, Montgomery curves were adopted for establishing secure communications over the Internet [4].

3) Edwards Curves: Montgomery curves have birationally equivalent Edwards versions which support the fastest (currently known) complete formulas for the elliptic-curve group operations. Let $\mathbb{F}_p$ be a prime field of odd characteristics; a twisted Edwards curve over $\mathbb{F}_p$ is defined as

$$E_{c,d}/\mathbb{F}_p : \alpha x^2 + y^2 = 1 + \beta x^2 y^2. \tag{3}$$

where $\alpha\beta(\alpha - \beta) \neq 0$.

Applications of Edwards curves to cryptography were developed by Bernstein and Lange [29]. Using a variant of Schnorr signature based on twisted Edwards curves, EdDSA [5] is a digital signature scheme with high performance, small signatures and public keys, which has gradually replaced ECDSA in the applications requiring digital signatures.

4) Target Curves: Our study covers common NIST curves and curves that meet all the requirements of the Safecurves website [30]. Better curve selection eliminates most attacks against curves, making a simple implementation more likely to be a secure one. Several different standards cover the selection of curves for use in ECC [3], [31]–[33]. Each of these standards tries to guarantee the difficulty of ECDLP but ignores ECC security. In order to avoid attacks that break real-world ECC without solving ECDLP, Daniel J. Bernstein and Tanja Lange designed the SafeCurves criteria [30] which tries to ensure both ECDLP security and ECC security.

B. Numerical Format of Floating-Point Numbers

Generally supported by a variety of platforms, the IEEE 754 standard [34] covers binary floating-point arithmetic and defines five rounding modes. The standard encodes binary floating-point data into three fields: the sign field (S), the biased exponent (E), and the fraction (F). The numerical value is represented as Equation (4)

$$(-1)^S \times (1 + F) \times 2^{E-bias}. \tag{4}$$

where *bias* is the exponent bias value. The 32-bit and 64-bit basic binary floating point formats defined in IEEE 754 [34] correspond to the C language data types `float` and `double`. This guarantees consistent computations across platforms and convenient exchange of data. The double precision floating point value used in this work has a 1-bit sign, an 11-bit exponent, and a 53-bit significand. The 53-bit significand includes an implicit integer bit of value 1 and an explicit

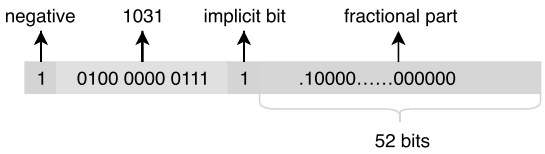


Fig. 1. "-384" represented in 64-bit basic binary floating point format.

TABLE I
DIFFERENCE BETWEEN FMA AND REGULAR MULTIPLICATION

$(1) A = 1 \times (1.00000000\ 00000000\ 0000001)_2$
$(2) B = -1 \times (1.00000000\ 00000000\ 0000010)_2$
$(3) \mathtt{rz}(A \times A + B) = 2^{-46}$
$(4) \mathtt{rz}(\mathtt{rz}(A \times A) + B) = 0$

52-bit fraction part. The bias is 1023. For example, the value -384 equals $(-1)^1 \times 2^8 \times 1.5$, can be represented as Fig. 1.

The format for 64-bit floating-point defined in IEEE 754 can't represent all real numbers for the sake of the limited length of bits. Hence, IEEE 754 specifies rounding modes to represent some values such as 2/3 in the format of floating point by losing a little precision. There are five rounding modes defined by IEEE 754 including *round-to-nearest (ties to even, ties away from zero)*, *round towards positive*, *round towards negative*, and *round towards zero*. Round towards zero (`rz`) which only retains the effective digit precision and directly discards the extra digits is the rounding mode used in our method.

C. The Fused Multiply-Add Operation

A fused multiply-add (FMA), included in IEEE 754 [34], can accelerate many computations and give more accurate results. IEEE 754 specifies that FMA must be performed with only one rounding step. This mechanism makes FMA more accurate than performing separate multiply and add operations with two rounding steps. Concretely, FMA completes $(x \times y) + z$ as $\mathtt{rz}((x \times y) + z)$ while without FMA the result of $(x \times y) + z$ is computed as $\mathtt{rz}(\mathtt{rz}(x \times y) + z)$. Here is an example in Table I that illustrates how the FMA operation works, using single precision values: $\mathtt{rz}(\mathtt{rz}(A \times A) + B)$ is computed separately by IEEE 754 multiply and IEEE 754 add, which loses all bits of precision, and the result is 0. Computing $\mathtt{rz}((A \times A) + B)$ by an IEEE 754 FMA provides a result equal to the expected mathematical value. As can be proved by this example, computing with FMA is more accurate than that with one multiply followed by one add.

Parallel platforms for accelerating scientific computation have complete support for FMA. NVIDIA with CUDA architecture complies with the IEEE 754 standard for binary floating-point arithmetic with acceptable deviations. CUDA CC 6.0 and CC 7.0 GPUs provide 64-bit binary DPF which can represent 53-bit integers. `__fma_rz (double x, double y, double z)`, one of the double precision intrinsics, computes $x \times y + z$ as a single operation in round-towards-zero mode.

III. DPF-ECC FRAMEWORK

The primitive used in ECC is point multiplication whose core computational overhead is modular multiplication.

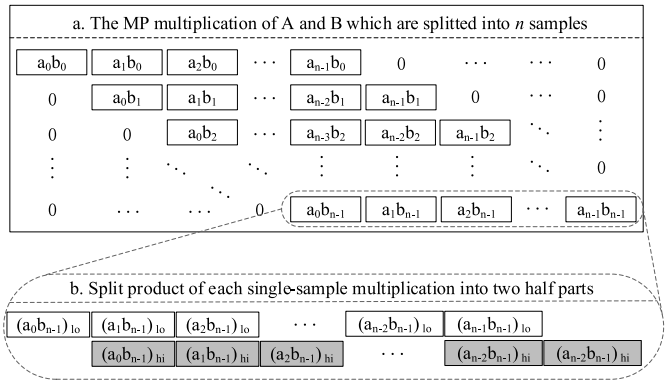


Fig. 2. The multiple precision multiplication of A and B.

This section first gives the overall DPF-ECC framework of efficient modular multiplication by using double precision floating computing power, and then elaborates on each subtask.

A. Decomposition of the Framework

The processor's digit size w is either 32 or 64, shorter than the size of the primes used in ECC. A common method for this scenario is multiple precision (MP) finite field arithmetic. The MP multiplication of A and B in $\mathbb{F}_p$ can be represented as $\sum_{i,j} a_i \cdot b_j$ as illustrated in Fig. 2 (a) where $(i + j)$ equals the column number.

For implementing MP modular multiplication of ECC with DPF computing power, a standard representation of elements in $\mathbb{F}_p$ is the baseline to regulate the input and output of finite field operations. The efficient design of single precision multiplication is the cornerstone of MP multiplication. Note that the result of MP multiplication is called the full product and is twice the width of the element in $\mathbb{F}_p$. The fast reduction can reduce the full product to the standard representation. Thus the proposed DPF-ECC framework for efficient modular multiplication contains four building blocks: the standardization of representation, single-sample multiplication, MP multiplication, and fast reduction.

B. The Standardization of Representation

A starting point to implement finite field arithmetic of ECC is to choose samples with a proper size for representing elements in $\mathbb{F}_p$.

Taking the special format of a DPF value into consideration, we regard the explicit 52-bit fraction part rather than the whole digital word size as its full radix. Theoretically, larger samples lead to higher parallelism and better performance. Whereas, in ECC schemes, representing integers with wider samples is not the only metric to implement modular multiplication. Reasonable selection of sample size can avoid overflow and skip trivial "simplification" steps.

ECC over different prime fields corresponds to the different optimal sample selection. Each integer in $\mathbb{F}_p$ should be evenly divided into several samples as much as possible. In order to avoid an extra alignment step of column accumulation in Fig. 2 (a), we represent the integer by several size-equal samples except for the last sample. We examine the DPF-ECC representation as follows.

Definition 3.1 (the DPF-ECC Representation): Let p be an k-bit number and the fraction size $w = 52$ be the full radix; Choose $\lambda, \mu \in (26, 52]$ and $\mu \leq \lambda$, define l satisfies $(l-1)\lambda + \mu = k$; then an element $A \in \mathbb{F}_p$ is represented by $(l-1)\lambda$-bit samples and a μ-bit sample as $(a_0, \ldots, a_{l-1})$, such that $A \equiv \sum_{i=0}^{l-1} a_i 2^{i\lambda} \mod p$, for $i \in [0, l-2]$, $0 \leq a_i < 2^{\lambda}$ and $0 \leq a_{l-1} < 2^{\mu}$.

We don't take $\lambda, \mu \leq 26$ into account, given that product of single-sample multiplication in the situation can be represented by one DPF value and has worse parallelism. The reason for defining $\mu \leq \lambda$ is given in Remark 3.2.

C. Single-Sample Multiplication

The underlying finite field arithmetic of point multiplication includes addition, subtraction, multiplication, etc., among which multiplication accounts for half of the operations and is the performance bottleneck of the computation. An efficient algorithm for single-sample multiplication is a basic module to implement multiple precision modular multiplication. Researchers have advised splitting the product of each single-sample multiplication into two half parts as shown in Fig. 2(b). In this subsection, we review two classical DPF-based multiplication algorithms and briefly give the reason why they are not the optimal choices for ECC.

1) Dekker's Algorithm: a famous problem dating back to the 1970s is the compute the error in a floating point multiplication. Dekker [35] solved this by splitting the two floating point values into halves and multiplying the halves. This results in a high product (essentially the product of the two original floating point values) and a low product (the error term). Dekker's method is very slow. With the advent of FMA hardware, there is a faster method to compute the high and low products as follows:

$$p\_hi = \_\_\texttt{fma\_rz}(\ x,\ y,\ 0.0). \tag{5}$$

$$p\_lo = \_\_\texttt{fma\_rz}(\ x,\ y,\ -p\_hi). \tag{6}$$

Dekker's algorithm does divide the product of two samples into the high product and the low product, and fits two parts to the significand of two DPF numbers. However, it is worth noting that the high and low products will not be conveniently "aligned" for the subsequent summation steps.

2) EZW Algorithm: Emmart *et al.* [18] utilized the 52-bit fraction of DPF numbers to represent integers in RSA. The algorithm sets the implicit leading 1 in the DPF significand so that the full 52 explicit bits are available for each half of the 104-bit products of samples. They obtained the high product from Equation (7). Since the product of two 52-bit samples is guaranteed to be less than 2^{104}, the significant bit of result in Equation (7) is 2^{104} and will become the implicit bit in binary representation. The next 52-bit fraction part will exactly match the 52-bit high product.

$$p\_hi = \_\_\texttt{fma\_rz}(\ x,\ y,\ 2^{104}). \tag{7}$$

EZW algorithm calculated the low product p_lo by Equation (6) which ensures that the significant bit of result in Equation (8) is 2^{52}. And the next 52-bit mantissa can match the 52-bit low product.

$$p\_lo = \_\_\texttt{fma\_rz}(\ x,\ y,\ 2^{104} + 2^{52} - p\_hi). \tag{8}$$

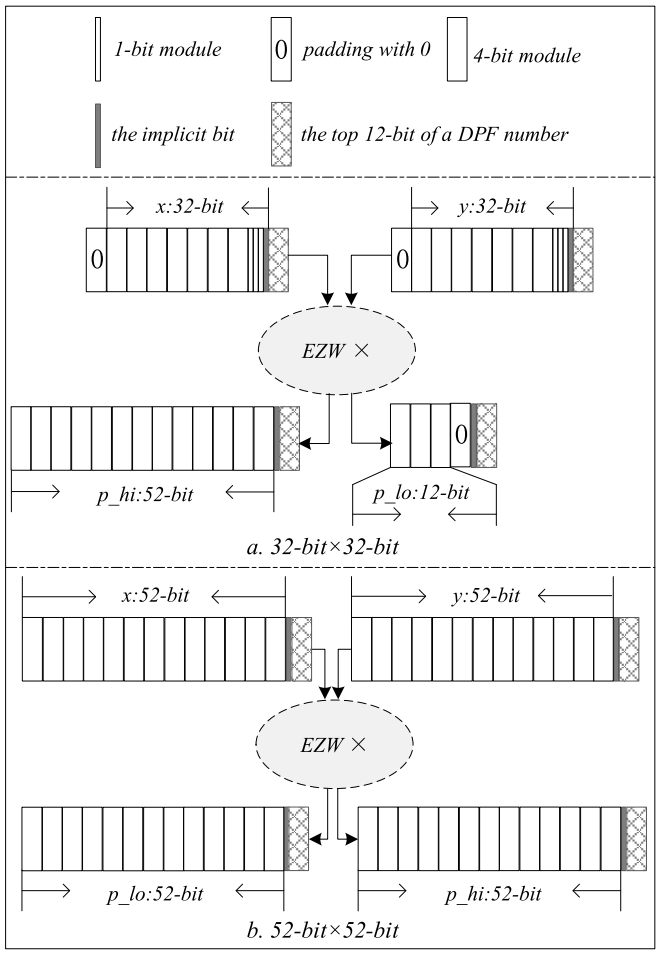


Fig. 3. Cases of DPF-based multiplication with EZW algorithm.

Fig. 3 illustrates the representation of products in the case of "$32-bit \times 32-bit$" and "$52-bit \times 52-bit$" with EZW algorithm. The EZW algorithm addresses the alignment problem, forcing the high and low products to be aligned to 52-bit boundaries and provides an elegant way to implement MP modular arithmetic with efficient DPF computing power for RSA. However, different from the RSA, generalized Mersenne primes widely used in ECC allow fast modular reduction algorithms to accelerate modular multiplication. Utilizing the 52-bit fraction part of DPF numbers to represent operands in ECC may not always gain the best performance.

3) Proposed Algorithm: In combination with DPF-ECC representation in Section III-B, we propose a more flexible single-sample multiplication algorithm for ECC. Specifically, we present the calculation of single-sample multiplication "$\lambda-bit \times \lambda-bit$", "$\lambda-bit \times \mu-bit$" and "$\mu-bit \times \mu-bit$".

Remark 3.1: Let x, y be two λ-bit samples. Leveraging with Equation (9) and Equation (10), we evenly split the product of "$x \times y$" into two DPF numbers, each preserving λ bits.

$$p\_hi = \_\_\texttt{fma\_rz}(\ x,\ y,\ 2^{\lambda+52}). \tag{9}$$

$$p\_lo = \_\_\texttt{fma\_rz}(\ x,\ y,\ 2^{\lambda+52} + 2^{52} - p\_hi). \tag{10}$$

Proof: The full product of two λ-bit samples is less than $2^{2\lambda}$, i.e. less than $2^{\lambda+52}$. Thus Equation (9) sets the significant implicit bit of mantissa to $2^{\lambda+52}$. And the 52-bit fraction part is composed of $(52-\lambda)$-bit zero (computed by $(52+\lambda)-2\lambda$) and the λ-bit high product. Equation (10) ensures the implicit bit of mantissa is 2^{52}. Then $(52-\lambda)$-bit zero and the λ-bit low product are concatenated as the fraction part. Fig. (4) shows the structure of the high sub-product and the low sub-product in Remark 3.1. □

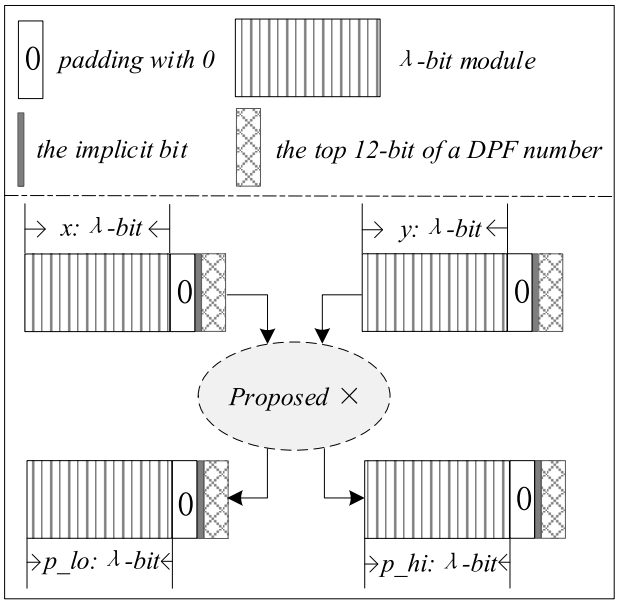


Fig. 4. Calculating two half products of $\lambda - bit \times \lambda - bit$ in remark 3.1.

The DPF-ECC representation involves not only the multiplication of "$\lambda - bit \times \lambda - bit$", but also the multiplication of "$\lambda-bit \times \mu-bit$" and the multiplication of "$\mu-bit \times \mu-bit$". How to deal with "$\lambda - bit \times \mu - bit$"? Is it necessary to introduce two extra parameters, that is, to replace λ with μ in Equation (9) and Equation (10) to calculate "$\mu-bit \times \mu-bit$"? Although only the last sample of the DPF-ECC representation is different from the others, we still need to consider whether the sub-products are aligned for subsequent accumulation.

Remark 3.2: Let x be a λ-bit sample and z be a μ-bit sample. For $\mu \leq \lambda$, the two half sub-products of single-sample multiplication "$x \times z$", and "$z \times z$" can be uniformly calculated by Equation (9) and Equation (10). There won't exist data misalignment in the sub-products to be accumulated.

Proof: Assume that $\mu \leq \lambda$, x is a λ-bit sample and z is a μ-bit sample. The product of "$x \times z$" is less than $2^{\lambda+\mu}$. Thus Equation (9) sets the significant implicit bit of mantissa to $2^{\lambda+52}$. And the 52-bit fraction part is composed of $(52-\mu)$-bit zero (computed by $(52+\lambda)$-$(\lambda+\mu)$) and the μ-bit high sub-product. Equation (10) ensures the implicit bit of mantissa is 2^{52}. Then the fraction part represents the λ-bit low sub-product. Similarly, the product of "$z \times z$" is divided into a $(52-2\mu+\lambda)$-bit high sub-product and a λ-bit low sub-product with Equation (9) and Equation (10) separately. Fig. 5 shows the structure of the sub-products concerning "$x \times z$" and "$z \times z$" in Remark 3.2. Note that the low sub-product of any form of single-sample multiplication is λ-bit in the case of $\mu \leq \lambda$. Only the samples of the rightmost edge and the bottom edge in Fig 2 representing the high sub-products have different

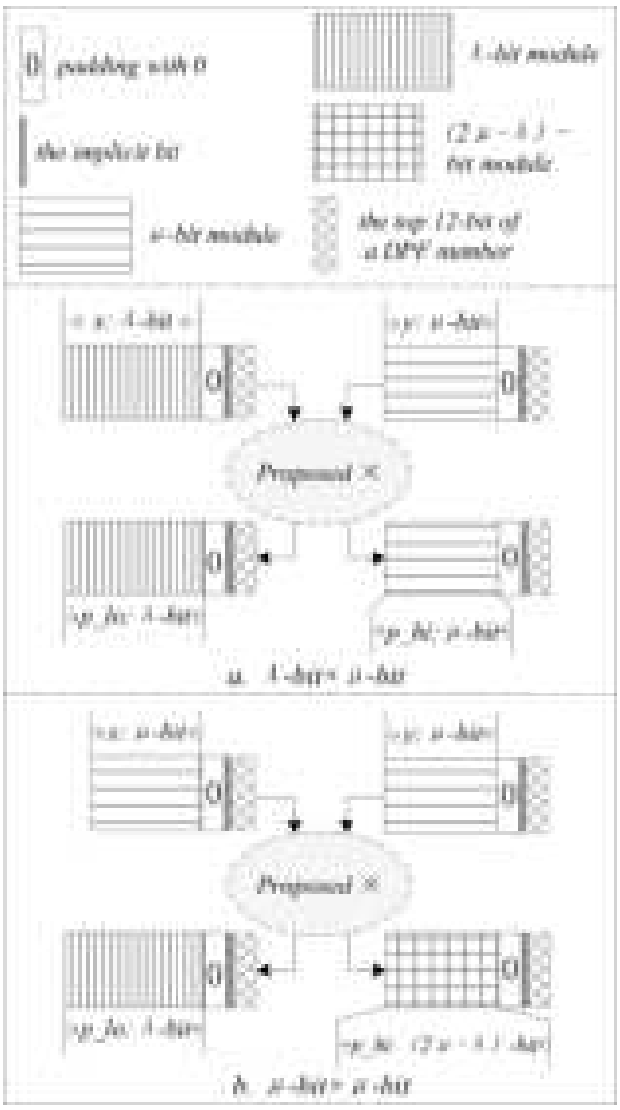


Fig. 5. The structure of the products concerning "$\lambda - bit \times \mu - bit$" and "$\mu - bit \times \mu - bit$" in remark 3.2.

sizes. Accordingly, Remark 3.2 avoids the misalignment of samples in subsequent summation. □

For $\mu > \lambda$, Equation (9) and Equation (10) won't solve all forms of single-sample multiplication. It is necessary to introduce additional parameters and perform additional data alignment operations to complete three types of single-sample multiplication, which brings extra performance overhead. Therefore, $\mu > \lambda$ is not taken into account. So far, the high sub-product and the low sub-product of any kind of single-sample multiplication can be stored separately in the fraction part of two DPF numbers. And the obtained sub-products can be directly used for subsequent accumulation.

D. Multiple Precision Multiplication

For two k-bit integers A and B represented by $(l-1)$ radix-2^λ samples and a radix-2^μ sample with $(l-1)\lambda + \mu = k$, we compute the full product of $A \times B$ by accumulating all the sub-products in each column as $C = \sum_{i=0}^{2l-1} c_i 2^{i\lambda}$. Notice that each c_i is a redundant sample, no more than $(2l-1)$ times of the radix-2^λ one (concluded from Fig. 2).

Since the top 12 bits are the sign field and the biased exponent, a DPF number only provides 53 bits of precision at most. In order to prevent overflow, we take advantage of the DPF binary representation and perform the summation in the 64-bit integer domain. We recover each half sub-product by masking the top 12 bits. Note that the sign bit of `p_hi` is always 0 and the exponent is $(52+\lambda)$ plus 1023 (the bias), thus, the top 12 bits of `p_hi` is always $0 \times 433 + \lambda$, and the top 12 bits of `p_lo` is always 0×433 (52 plus 1023). We can obtain the correct results by initializing the column sums with the correct negative values. So far, we have recovered $2l$ summation columns without any interference bits in the integer domain.

Algorithm 1 Multiple Precision Multiplication with FMA

Input:

Two integers A and B are in corresponding DPF-ECC representations, i.e., each of the least significant $l-1$ samples has λ bits and the most significant sample has μ bits.

Output:

The full product C represented by $2l$ `uint64_t` numbers, and each sample of C is redundant.

1: $c1 = 2^{52+\lambda}$, $c2 = 2^{52+\lambda} + 2^{52}$
2: **for** $i = 0$ *to* $l-1$ **do**
3: $C_i = i \times (0 \times 433 + \lambda) + (i+1) \times 0 \times 433$
4: $C_i = -(C_i \& 0xFFF) \ll 52$
5: $C_{2l-1-i} = (i+1) \times (0 \times 433 + \lambda) + i \times 0 \times 433$
6: $C_{2l-1-i} = -(C_{2l-1-i} \& 0xFFF) \ll 52$
7: **end for**
8: **for** $i = 0$ *to* $l-1$ **do**
9: **for** $j = 0$ *to* $l-1$ **do**
10: $p\_hi = \_\_\texttt{fma\_rz}(a_j, b_i, c1)$
11: $C_{i+j+1} = C_{i+j+1} + \texttt{conv\_2\_u64}(p\_hi)$
12: $sub = c2 - p\_hi$
13: $p\_lo = \_\_\texttt{fma\_rz}(a_j, b_i, sub)$
14: $C_{i+j} = C_{i+j} + \texttt{conv\_2\_u64}(p\_lo)$
15: **end for**
16: **end for**

Combined with Fig. 2 and FMA, Algorithm 1 computes an l-sample by l-sample multiple precision full product returning $2l$ column sums of the result. The double loops (Line 8-16) in the algorithm show the cumulative process. The function `conv_2_u64` (Line 11 and Line 14) converts a `double` representation to a `uint64_t` representation. Because the length of a sign bit and 11 exponent bits is invariable, we could easily intercept the mantissa of each sample in the format of `uint64_t`. The algorithm also contains of a routine loop (Line 2-7) which generates the appropriate initial value for the column sum that will eliminate the $(0 \times 433 + \lambda)$s and 0×433s. Since the effective part of each sample is λ bits, and $(64-\lambda)$ bits are left, we can add up to less than $2^{64-\lambda}$ product terms in a column and ignore carries generated by the $(0 \times 433 + \lambda)$s and 0×433s. After Algorithm 1, the full product can be represented as $2l$ redundant `uint64_t` samples.

E. Fast Reduction Suitable for DPF-ECC

Standardization of the output is necessary because the output of one MP multiplication may serve as the input to another operation. This subsection describes the workflow of fast reduction for reducing $2l$ redundant samples to a standard DPF-ECC representation with l samples.

It has long been known that generalized Mersenne primes widely used in ECC allow fast modular reduction algorithms. Generalized Mersenne primes include Crandall primes and Solinas primes. Crandall [36] used shifts and adds only instead of divisions to accomplish modular arithmetic of Crandall primes. Solinas [37] generalized this technique to a larger class of primes. The methodologies presented in [36] and [37] are generally referred to as fast reduction algorithm, which are well suited for machine implementation. Because the DPF-ECC framework adopts a novel integer representation method specific to DPF numbers, the fast reduction algorithm mentioned above cannot be directly ported to this framework. Therefore, we make adjustments on the basis of the traditional fast reduction algorithm and give a reduction strategy suitable for DPF-ECC framework.

We first examine the basic idea of the fast modular reduction algorithm in [36] and [37]. For $A, B \in \mathbb{F}_p$ where p can be represented by at least k bits and A, B are both in the form of the DPF-ECC representation, the multiple precision full product $C = \sum_{i=0}^{2l-1} c_i$ of A and B has $2l$ column sums. Let T be the integer represented by the k most significant bits of C, and V be the k least significant bits of C, thus

$$C = 2^k \cdot T + V. \tag{11}$$

with T and V each being k bits integers [38]. Then

$$C \equiv (2^k - p) \cdot T + V \pmod{p} \tag{12}$$

As can be seen from Equation (12) that modular reduction can be replaced by one multiplication and one addition as long as p is close enough to 2^k and T is small enough. In order to facilitate the calculation combined with DPF-ECC representation, we convert Equation (12) to Equation (13). The first l samples have $l \cdot \lambda$ least significant bits of the full product. For $\mu \leq \lambda$ and $(l-1)\lambda + \mu = k$, we should multiply $2^k - p$ with $2^{(l \cdot \lambda - k)}$ as a coefficient instead of just taking $2^k - p$ as a coefficient.

$$\begin{aligned} C &\equiv (2^k - p) \cdot 2^{(l \cdot \lambda - k)} \cdot \sum_{i=0}^{l-1} c_{i+l} 2^{i\lambda} + \sum_{i=0}^{l-1} c_i 2^{i\lambda} \pmod{p} \\ &\equiv (2^k - p) \cdot 2^{(\lambda - \mu)} \cdot \sum_{i=0}^{l-1} c_{i+l} 2^{i\lambda} + \sum_{i=0}^{l-1} c_i 2^{i\lambda} \pmod{p} \end{aligned} \tag{13}$$

In most cases, it won't produce the standard DPF-ECC representation which has l samples by performing the fast reduction only once. However, in terms of our target ECCs of which p is a generalized Mersenne prime, extra samples of C can be cleared by re-applying the fast reduction.

After accumulating sub-products, the full product $C = \sum_{i=0}^{2l-1} c_i 2^{i\lambda}$ has $2l$ redundant samples where the upper $(64 - \lambda)$ bits of each term overlap with the next summation term. The intermediate result of C obtained after each round of fast reduction is redundant as well.

Definition 3.2 (the Simplification Step): For $C = \sum_{i=0}^{t-1} c_i 2^{i\lambda}$ composed of $t \in [l+1, 2l]$ redundant samples, divide each column term into two parts, the lower λ bits and the upper $(64 - \lambda)$ bits. The upper $(64 - \lambda)$ bits will be cleared to zero after getting added to the left of the next column (Equation (14), where $\boldsymbol{mask}_\lambda$ is $2^\lambda - 1$).

$$c_{i+1} = c_{i+1} + (c_i \gg \lambda),\ c_i = c_i\ \&\ \boldsymbol{mask}_\lambda,\ i \in [0, t-2]. \tag{14}$$

Then each of the first $(t-1)$ terms is definitely within λ bits and only the *t-th* term is redundant.

The "simplification" step is implemented before each round of fast reduction in order to avoid overflow. However, we can skip this step in Remark 3.3.

Remark 3.3 (Skip the "Simplification" Step): The MP modular multiplication in $\mathbb{F}_p$ can skip the "simplification" step as long as each term of intermediate results is less than 2^{64} after fast reduction.

The full product has been reduced to $\hat{C} = \sum_{i=0}^{l-1} \hat{c}_i 2^{i\lambda}$ after several rounds of fast reduction. Next, we need to further reduce $\hat{C}$ and simplify the samples. It's similar to the "simplification" step.

Definition 3.3 (Carry-only Mode): Each term of $\hat{C}$ is 64 bits, rather than λ bits. Equation (15) makes each of the first $(l-1)$ terms a λ-bit sample. Only the l-th term is redundant. We regard the top $(64 - \mu)$ bits of c_{l-1} as the carry.

$$\hat{c}_{i+1} = \hat{c}_{i+1} + (\hat{c}_i \gg \lambda),\ \hat{c}_i = \hat{c}_i\ \&\ \boldsymbol{mask}_\lambda,\ i \in [0, l-2]. \tag{15}$$

We eliminate the top $(64 - \mu)$ bits of the l-th sample $\hat{c}_{l-1}$ as follows.

Remark 3.4 (the Resolution of Carry): We multiply the carry with $2^{(\lambda-\mu)} \cdot (2^k - p)$ first, then zero the top $(64 - \mu)$ bits of $\hat{C}$, finally add the product to $\hat{C}$. Then we convert $\hat{C}$ to carry-only mode again. If the carry still exists, we continue with this step to gradually reduce the carry. Find the max times of carry reduction for p to be performed, then the resolution of carry can be implemented in constant time.

$\hat{C}$ after the resolution of carry is in the format of DPF-ECC representation and can be used in the next round of MP modular multiplication.

IV. Case Study

In this section, we show how the DPF-ECC framework can be applied to the target ECCs of Section II-A.4. The field of target curves is generalized Mersenne prime field which can be categorized as Crandall primes [36] and Solinas primes [37]. Crandall primes that are in the form of $2^k - \sigma$ where σ is a small odd number (let $\sigma < 2^\lambda$). Solinas primes $(2^k - 2^t \pm \cdots \pm 1)$ are primes numbers of form $f(2^n)$, where $f(x)$ is a low-degree polynomial with small integer coefficients.

We start case studies with parameter selections. In combination with Section III-B, we choose the parameters that represent the elements of the prime field according to the following four principles: 1) l is as small as possible. 2) $\mu \leq \lambda$. 3) $(l-1)\lambda + \mu = k$. 4) μ is close to λ.

❶ l is as small as possible. The element of prime field $\mathbb{F}_p$ is represented by l samples. More samples mean more calculation steps. In particular, the increase in multiplication

TABLE II
ELLIPTIC CURVES IN FORM OF $2^k - \sigma$

Curve	p	Equation	Shape	Parameters						Satisfy Equ. (17)?
				k	σ	λ	μ	l	$\lceil \frac{p}{52} \rceil$	
M-221 [39]	$2^{221}-3$	$y^2 = x^3 + 117050x^2 + x$	Montgomery	221	3	45	41	5	5	✓
E-222 [39]	$2^{222}-117$	$x^2 + y^2 = 1 + 160102x^2y^2$	Edwards	222	117	45	42	5	5	✓
Curve1174 [40]	$2^{251}-9$	$x^2 + y^2 = 1 - 1174x^2y^2$	Edwards	251	9	51	47	5	5	✓
Curve25519 [41]	$2^{255}-19$	$y^2 = x^3 + 486662x^2 + x$	Montgomery	255	19	51	51	5	5	✓
E-382 [39]	$2^{382}-105$	$x^2 + y^2 = 1 - 67254x^2y^2$	Edwards	382	105	48	46	8	8	✓
M-383 [39]	$2^{383}-187$	$y^2 = x^3 + 2065150x^2 + x$	Montgomery	383	187	48	47	8	8	✓
M-511 [39]	$2^{511}-187$	$y^2 = x^3 + 530438x^2 + x$	Montgomery	511	187	52	43	10	10	✗
Curve41417 [42]	$2^{414}-17$	$x^2 + y^2 = 1 + 3617x^2y^2$	Edwards	414	17	52	50	8	8	✓
NIST P-521 [3]	$2^{521}-1$	$y^2 = x^2 - 3x + b$	Short Weierstrass	521	1	48	41	11	11	✓

equals the square of the increase in l. A small l can improve the parallelism of the algorithm and decrease the number of DPF register consumption. Since the full-radix of the DPF-ECC framework is 52-bit, we choose l as close as $\lceil \frac{p}{52} \rceil$.

❷ $\mu \leq \lambda$. As illustrated in Section III-C, the single-sample multiplication can be uniformly calculated by Equation (9) and Equation (10) with only two parameters $2^{\lambda+52}$ and $2^{\lambda+52} + 2^{52}$. If $\mu > \lambda$, new parameters may need to be introduced to calculate single-sample multiplication, and the sub-products of "$\mu \times \lambda$" and "$\mu \times \mu$" are unaligned with "$\lambda \times \lambda$". Additional alignment operations are required. There are also special cases where $\mu > \lambda$ require the same number of parameters as $\mu \leq \lambda$ to perform single-sample multiplication and don't result in unaligned data, which we will discuss in detail later.

❸ $(l-1)\lambda + \mu = k$. This principle guarantees the complete representation of the elements in $\mathbb{F}_p$.

❹ μ is close to λ. The similarity of μ and λ is beneficial to skipping the "simplification" step in Section III-E.

A. Crandall Primes

Table II lists the elliptic curves that use Crandall primes ($p = 2^k - \sigma$) and corresponding parameter selections. The fast reduction Equation (13) can be specified as Equation (16) for Crandall primes.

$$C \equiv \sigma \cdot 2^{\lambda-\mu} \cdot \sum_{i=0}^{l-1} c_{i+l} 2^{i\lambda} + \sum_{i=0}^{l-1} c_i 2^{i\lambda} \ (mod\ p) \quad (16)$$

The full product $C = \sum_{i=0}^{2l-1} c_i 2^{i\lambda}$ of MP multiplication of two elements in $\mathbb{F}_p$ has $2l$ summation terms as described in Section III-D. For the general case where $\mu \leq \lambda$, possessing the most cumulative entries, the l-th term c_{l-1} accumulates $(2l-1)$ sub-products. Note that each sub-product is less than 2^λ. For $\forall\, i \in [0, 2l-1]$, $0 \leq c_i \leq (2l-1) \times (2^\lambda - 1)$ and each redundant sample c_i is represented by a 64-bit integer. Thus we could skip the "simplification" step for $p = 2^k - \sigma$ if the maximum value of term after fast reduction in Equation (17) (the left part) is less than 2^{64}.

$$\begin{aligned} &\sigma \times 2^{(l\times\lambda-k)} \times (2l-1) \times (2^\lambda - 1) + (2^\lambda - 1) < 2^{64} \\ &\quad \Longrightarrow\ (\sigma \times 2^{\lambda-\mu} \times (2l-1) + 1) \times (2^\lambda - 1) < 2^{64} \quad (17) \end{aligned}$$

Parameters of all curves except M-511 presented in Table II can satisfy Equation (17), which reduces the overhead by skipping the "simplification" step. In addition, since l is always equal to $\lceil \frac{p}{52} \rceil$, the choice of these parameters also maximizes the parallelism of finite field operations.

TABLE III
ELLIPTIC CURVES IN THE FORM OF $f(2^n)$

Curve	p	Parameters			
		λ	μ	l	$\lceil \frac{p}{52} \rceil$
NIST P-192 [3]	$2^{192} - 2^{64} - 1$	48	48	4	4
NIST P-224 [3]	$2^{224} - 2^{96} + 1$	45	44	5	5
NIST P-256 [3]	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	52	48	5	5
NIST P-384 [3]	$2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$	48	48	8	8
Curve448 [43]	$2^{448} - 2^{224} - 1$	50	48	9	9

Based on the above discussion, the full product C in $\mathbb{F}_p$ can be reduced to the carry-only mode $\hat{C} = \sum_{i=0}^{l-1} \hat{c} 2^{i\lambda}$ with only one round fast reduction. However, each term of $\hat{C}$ is still redundant, rather than a standard sample.

Remark 4.1 (Eliminate Carries within Two Rounds.): The result $\hat{C}$ of fast reduction of MP multiplication in $\mathbb{F}_p$ presented in Table II can be reduced within k bits in two rounds of carry resolution.

Proof: Equation (15) is applied to convert the $\hat{C}$ into Carry-only Mode. Let $\hat{carry}$ be the carry in the Carry-only Mode. We perform the resolution of carry with Equation (18) and Equation (19):

$$\hat{c}_0 += (\hat{c}_{l-1} \gg \mu) \cdot \sigma \quad (18)$$

$$\hat{c}_{l-1} = \hat{c}_{l-1} \& mask_\mu \quad (19)$$

After the first round of carry resolution, the range of $\hat{C}$ is

$$\hat{C} \in [0, 2^k + \sigma \cdot 2^{64-\mu}). \quad (20)$$

The value is still not in the range of $[0, 2^k - 1]$. The $\sigma \cdot 2^{64-\mu}$ computed from Table II is small enough ($< 2^{29}$) that $\hat{C}$ is less than 2^{k+1} but has 1-bit carry at most. With Equation (15), Equation (18) and Equation (19) continue the second round of carry resolution. Then

$$\hat{C} \in \begin{cases} [0, 2^k - 1] & if \quad \hat{carry} = 0 \\ [\sigma, \sigma \cdot 2^{64-\mu} + \sigma) & if \quad \hat{carry} = 1. \end{cases} \quad (21)$$

Thus the result of $\hat{C}$ belongs to $[0, 2^k - 1]$ after two rounds of carry resolution. Obviously, $\hat{c}_0$ becomes redundant again. Equation (15) can make $\hat{C}$ a standard DPF-ECC format. □

B. Solinas Primes

Besides Crandall primes, parameters of Solinas primes used for ECC are listed in Table III. They are prime numbers that

have the form $f(2^n)$, where $f(x)$ is a low-degree polynomial with small integer coefficients. The form of them is not as concise as that of Crandall primes, but in this case, the $2l$-sample full product C of MP multiplication can still be reduced to l samples by the fast reduction algorithm.

We present the reduction methodology on NIST P-256 and NIST P-384 in Appendix A and Appendix B respectively. Here gives the reduction process of Curve448 in detail as an example. Note that $p = 2^{448} - 2^{224} - 1$, which implies $2^{448} \equiv 2^{224} + 1 \ (mod\ p)$. Thus $C \equiv \sum_{i=0}^{8} c_i \cdot 2^{50i} + \sum_{i=9}^{17} c_i \cdot 2^{50i} \ (mod\ p)$ can be transformed into

$$\begin{aligned} C \equiv & \sum_{i=0}^{8}(c_i + 4c_{i+9}) \cdot 2^{50i} + \sum_{i=4}^{8} 2^{26} \cdot c_{i+5} \cdot 2^{50i} \\ & + \sum_{i=0}^{3} 2^{28} \cdot c_{i+14} \cdot 2^{50i} + \sum_{5}^{8} 4 \cdot c_{i+9} \cdot 2^{50i} \ (mod\ p). \end{aligned} \tag{22}$$

As shown in Equation (23), instead of just multiplying by the terms of 2^{26} or 2^{28}, we use the `&` and `shift` operations to prevent the overflow.

$$\begin{aligned} C \equiv & \sum_{i=0}^{8}(c_i + 4c_{i+9}) \cdot 2^{50i} + \sum_{i=4}^{4}[(c_{i+9} \gg 24) \ll 26] \cdot 2^{50i} \\ & + \sum_{i=4}^{8}[(c_{i+5} \& mask_{24}) \ll 26] \cdot 2^{50i} + (c_{13} \gg 24) \cdot 4 \\ & + \sum_{i=5}^{8}(c_{i+4} \gg 24 + 4c_{i+9}) \cdot 2^{50i} + \sum_{i=1}^{4}(c_{i+13} \gg 22) \cdot 2^{50i} \\ & + \sum_{i=0}^{3}[(c_{i+14} \& mask_{22}) \ll 28] \cdot 2^{50i} \ (mod\ p) \end{aligned} \tag{23}$$

Every term of C in Equation (23) is less than 2^{64}. Just like the Remark 4.1 for Crandall primes, we can also settle the carry with two rounds carry resolution for ECCs listed in Table III. The process of proof is similar to that in Section IV-A with Equation (18), (19), (20) and (21). It is easy to conclude that the DPF-ECC framework is applicable to curves in Table III as well.

C. Discussion

The techniques for optimizing implementations of curves are further discussed here.

We don't have to reduce the result strictly within k bits (until we get the final result of point multiplication), a $(k+1)$-bit number is just fine as the intermediate value, and it brings the following two benefits:

Firstly, it can reduce the computational cost by using just one round of carry resolution (by executing Equation (18) and Equation (19) sequentially). With only one round of carry resolution, the result can fall into the range of $[0, 2^{k+1})$. In this way, the output can directly serve as the input of the next function (modular multiplication, addition and subtraction, etc.).

Secondly, extending the range of the result to $(k+1)$ bits leads to the length of the most significant sample being $\mu + 1$. If $\mu < \lambda$, then $\mu + 1 \leq \lambda$, and the extra bit of the most significant samples doesn't introduce extra overhead for the multiplication as described in Section III-C. The only exception where $\mu = \lambda$ is $p = 2^{255} - 19$. The most significant sample is 52 bits when extending the range of the result to 256 bits for $p = 2^{255} - 19$. Thus the single sample multiplication involves "51-bit×51-bit", "51-bit×52-bit" and "52-bit×52-bit". The low product and high product of these three kinds of single sample multiplication can be uniformly calculated by Equation (24) and Equation (25).

$$p\_hi = \_\_fma\_rz(x, y, 2^{103}). \tag{24}$$

$$p\_lo = \_\_fma\_rz(x, y, 2^{103} + 2^{52} - p\_hi). \tag{25}$$

Here we elaborate on the calculation:

- The high product of "51-bit×51-bit" and "51-bit×52-bit". In this situation, the products are guaranteed to be less than 2^{103}, if we compute $p\_hi = a_i \cdot b_i + 2^{103}$, then the 2^{103} will be the most significant bit of the result and will be the implicit bit in the DPF binary representation. For "51-bit×51-bit", the next 52 bits (the explicit bits) of the mantissa is made up of 0 and the 51-bit high product. For "51-bit×52-bit", the next 52 bits of the mantissa will exactly match the 52-bit high product.
- The low product of "51-bit×51-bit" and "51-bit×52-bit". Likewise, computing $p\_lo = a_i \cdot b_i + (2^{103} + 2^{52}) - p\_hi$ ensures that 2^{52} will be the implicit bit of p_lo and the next 52 explicit bits are made up of 0 and the 51-bit low product.
- "52-bit×52-bit" can be classified as "51-bit×52-bit". Each redundant sample is up to $(7 \times (2^{51} - 1) + 2 \times (2^{52} - 1))$. By accumulating 19 times of c_{i+5} to c_i, each c_i $(i \in [0, 4])$ is up to

$$\begin{aligned} & 19 \times [7 \times (2^{51} - 1) + 2 \times (2^{52} - 1)] + (2^{51} - 1) \\ & \quad < 2^{59} - 1. \end{aligned} \tag{26}$$

With only one round of simplification, the result can fall into the range of $[0, 2^{256})$, because:

$$19 \times (2^{59-51} - 1) + 2^{255} - 1 < 2^{255} + 4845 < 2^{256} - 1. \tag{27}$$

There only exists one product term (multiply the most significant sample by the most significant sample) that involves "52-bit×52-bit". Since the operands of multiplication are always small enough ($< 2^{255} + 4845$), the whole product ($< (2^{255} + 4845)^2$) is less than 2^{511}, which implies "52-bit×52-bit" won't exceed 103 bits.

Therefore, the modular multiplication over $\mathbb{F}_p$ where $p = 2^{255} - 19$ also introduces only two parameters, and does not require additional operations of alignment during the summation of sub-products.

V. Experiments

In this section, we use the DPF-ECC framework to implement the primitive of key exchange in RFC7748 [4] and the primitives of digital signature in RFC8032 [5]. RFC7748 and RFC8032 respectively recommend Curve25519/448 for Diffie-Hellman key exchange protocol and Edwards25519/448 for

digital signature algorithm, which have gained great momentum in recent years. NIST also planned to approve the Curve25519/448 and Edwards25519/448 in the upcoming FIPS PUB 186-5. The two prime numbers involved in Curve25519/448 and Edwards25519/448 are representatives of Section IV.

We discuss the advantages of the proposed DPF-ECC framework, especially in terms of performance, by implementing the above popular primitives.

A. Implementation Details

The specific primitives for implementation and evaluation include the point multiplication of Curve25519/448 for key exchange, the known point multiplication of Edwards25519/448 for signing, the unknown point multiplication of Edwards25519/448 for signature verification. In this subsection, we state the implementation details and optimization tips.

1) Curve25519/448: We implement point multiplication of Curve25519/448 as illustrated in [4]. The inputs and outputs of Curve25519/448 which are x-coordinate-only algorithms are both in the representation form described in Section IV. To accelerate implementation, we simplify computing steps and reuse the variables as much as possible. We also implement the function such that it runs in constant time.

2) Edwards25519/448: Assume that $\boldsymbol{G}$ is a known point, with the portion-by-portion addition method described in [14], we split the scalar d into n groups to generate a pre-computed table and compute $[d]\boldsymbol{G}$, i.e. $d = d_{n-1}|\ldots|d_1|d_0$. We define $m = \frac{k}{n}$. Each group contains 2^m points, and one point is composed of 2 (the number of coordinates) variables. Thus, $[d]\boldsymbol{G} = \sum_{i=0}^{n-1} d_i 2^{im}\boldsymbol{G}$. Similar to the standard representation in Section IV, we respectively use l `uint64_t` numbers to represent the intermediate result of Edwards25519/448. When $m = 32$, the size of the offline pre-computed table is much larger than the global memory of GPU. As to $m = 16$, the size of the table for known point multiplication of Edwards25519/448 is feasible.

We use the fixed window method [44] which scans certain multiple (ω) bits every time to complete unknown point multiplication of Edwards25519/448. ω denotes the `window size`. First, we generate an online table consisting of 2^ω points for each thread on-the-fly. And the 2^ω points are $[0]\boldsymbol{P}, [1]\boldsymbol{P}, [2]\boldsymbol{P}, \ldots [2^\omega - 1]\boldsymbol{P}$ where $\boldsymbol{P}$ denotes the unknown point. Note that the online tables store extended twisted Edwards coordinates (X, Y, Z, T) [45] to avoid the expensive cost in point doubling operations. To maximize parallelism, each thread conducts one ECC implementation. Thus the table size for each thread is $(2^\omega \times l \times 64 \times 4)$ bits. After rigorous tests, $\omega = 4$ is the optimal window size and the size of the pre-compute tables for all threads is $T \times \frac{l}{2}$ KB, where T is the total number of threads.

B. Performance Evaluation

We present our experiments setup, discuss the experimental performance in this subsection.

1) Experiment Setup: We use CUDA to program our codes on Linux operating system. And an NVIDIA GPU is used for evaluation. The specific hardware and software configurations in the experiment are detailed in Table IV. The data source in experiments is generated by linux pseudorandom number generator "/dev/urandom".

TABLE IV
TARGET PLATFORM CONFIGURATION

OS	Ubuntu 16.04
CUDA Toolkit	CUDA 10.0
CPU	Intel Core i7-7700K CPU @ 4.20GHz
GPU	Tesla P100-PCIE-16G

TABLE V
REGISTERS PER THREADS

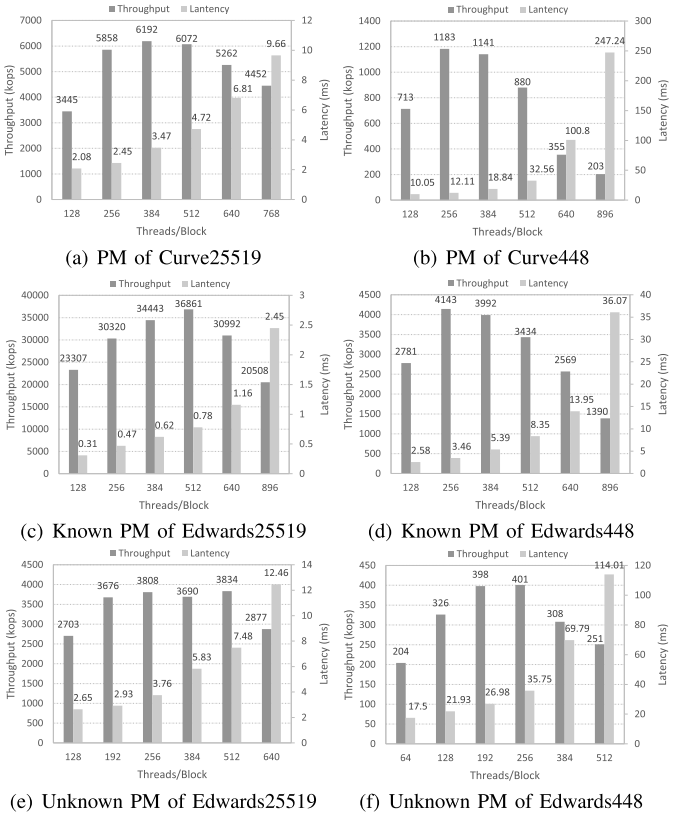
Threads/Block	≤ 256	384	512	640	768	896
Regs/Thread	256	168	128	96	80	72

2) Performance Tunning: Latency and throughput which represents operations of ECC per second (ops) are critical indicators for performance evaluation of our implementation. Designed for high-concurrency scenarios, we set the goal as "as large a throughput with acceptable latency", and thus the latency metric indicates how practical the high-throughput solution is. The best threads per block combination for throughput-latency mainly depends on the specifics of each architecture, which requires its own optimization. Here are several GPU's dynamic configuration parameters that may affect the performance of the kernel during experiments:

- *Batch Size*: the number of point multiplications per GPU kernel launch.
- *Threads/Block*: the number of CUDA threads contained in a CUDA block.
- *Regs/Thread*: the maximum number of registers assigned for each CUDA thread.

Making full use of the total available threads can keep the pipeline busy most of the time and take greater advantage of the computing power in GPUs. Each thread runs one ECC example. Thus, *Batch Size*, the product of the number of blocks and *Threads/Block*, plays a critical role in the performance of GPU kernels. Nevertheless, limited by the amount of resources, the overall throughput will drop off when the number of threads per block exceeds a reasonable value.

For maximum performance, we fix the number of blocks for Tesla P100 to 56. The number of 32-bit registers per CUDA block is 65536 to which the configuration *Regs/Thread* is restricted. After adjusting and testing with different Threads/Block, we give the optimal amount of registers assigned to each thread in Table V. We demonstrate the results of experiments and the impact of batch size on performance on selected GPUs in Fig. 6. All experimental results are labeled above the column chart. Figures indicate that a larger batch size makes higher throughput and little increment in latency within a certain range. However, a larger batch size also requires more resources. Typically, in the situation of shortage of registers, variables spill into off-chip local memory, which is hundreds of times slower than registers. When the resource demand far exceeds the threshold and surpasses the benefit of larger batch size, performance begins to decline. Table VI summarizes the peak performance of each primitive on Tesla P100.



(a) PM of Curve25519 (b) PM of Curve448

(c) Known PM of Edwards25519 (d) Known PM of Edwards448

(e) Unknown PM of Edwards25519 (f) Unknown PM of Edwards448

Fig. 6. Throughput and latency of experiments on Tesla P100 (the number of blocks is 56).

TABLE VI
PEAK PERFORMANCE OF PRIMITIVES ON TESLA P100

Primitives	Shape	Throughput (kops)	Latency (ms)
Unknown PM	Curve25519	6,192	3.47
	Curve448	1,183	12.11
Known PM	Edwards25519	36,861	0.78
	Edwards448	4,143	3.46
Unknown PM	Edwards25519	3,808	3.76
	Edwards448	398	26.98

3) Impact of Skipping "Simplification" on Performance: In this part, we focus on the impact of skipping "simplification" (see details in Section III-E) on performance.

There is no doubt that full use of DPF resources in itself has the greatest improvement in performance. However, the skipping "simplification" in fast reduction also improves performance considerably. As illustrated in Fig 7, the skipping "simplification" brings 12.9%, 5.4%, 16.9% throughput performance improvement to known and unknown point multiplication of Edwards25519, unknown point multiplication of Curve25519 at their peak points on Tesla P100, respectively, compared with the "Non-Simplification" method.

C. Related Work Comparison

In this section, we compare our work with the floating-point-based implementation, integer-based implementations, respectively.

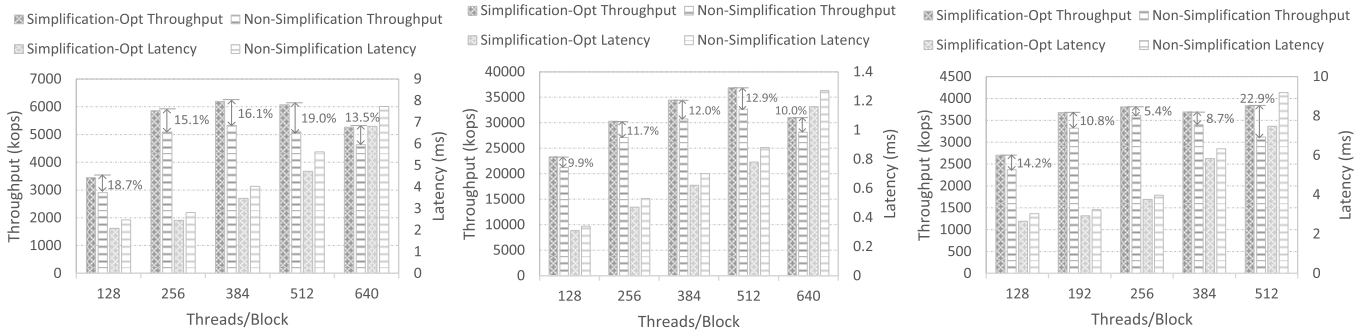
1) vs. Floating-Point-Based Implementation on GPU: Bernstein *et al.* [12] (Eurocrypt '09) employed floating-point arithmetic to implement stage-1 ECM. For a fair comparison, we scale their performance in Table VIII. Taking the difference of ECC key sizes and GPU platforms into consideration, we firstly multiply their throughput by the ratio of GFLOPS, $\frac{8,705.5}{1,192.3}$ where 8,705.5 is the average of 8,071 and 9,304. Note that work [12] implemented point multiplication of a 280-bit Edwards Curve over a generic modulus by a 11797-bit scalar, thus we further multiply its performance with $\frac{11,797}{k} \times (\frac{280}{k})^2 \times 2$, where "$\times 2$" represents the double computation cost for Montgomery multiplication over a generic modulus compared with the fixed modulus in this paper.

Even considering the difference of platforms and curves, our implementation achieves over 19 times speedup over [12]. Part of the performance advantage comes from the use of DPF power of GPU. Whereas, Bernstein *et al.* [12] only exploited single precision floating point power of GPU. What's more, we fully utilize each thread in a warp by performing all computations in one thread, while they used only 28 threads in a warp. Furthermore, they used Separated Operand Scanning (SOS) Montgomery multiplication method which is inefficient in contrast to our simple scheme. Last but not least, they performed all computations with floating-point power, whereas, some operations (e.g., addition, subtraction) performed with integer instructions are more efficient. We combine DPF power and integer instructions in a new computing method to save unnecessary costs.

2) vs. Integer-Based Implementations on GPU: Many papers have implemented ECC schemes with integer computing power.

Ant$\widetilde{a}$o *et al.* [46], [47] and Pu and Liu [48] relied on the Residue Number System (RNS) to extract parallelism on high precision integer arithmetic for elliptic curve point multiplication. With the operands sharing a common RNS representation, the computation can be performed in parallel on the corresponding residues. However, the transformation between large integers and RNS representations consumes too many instructions. Several previous works [11], [12], [26] completed the Montgomery multiplication using a single thread in GPUs for ECC. Montgomery multiplication is a kind of modular multiplication algorithm, which is widely implemented in GPUs. Pan *et al.* [14], Zheng *et al.* [13], Bos [49] and Szerwinski and Guneysu [11] utilized fast reduction to complete modular multiplication of ECC over generalized Mersenne prime fields in one thread. In 2017, Pan *et al.* [14] utilized GTX 780Ti to implement NIST P-256 signature verification as an efficient ECC server which reached 920,000 operations per second (ops). Based on GTX TITAN, Mahe and Chauvet [50] implemented Curve25519 and NIST Curves. In 2018, Dong *et al.* [15] scheduled and optimized Curve25519 with CUDA parallel thread execution (PTX) instruction set architecture (ISA) instructions on GTX TITAN and GTX 1080.

Table VII shows the performance comparison between integer-based point multiplication of Curve25519 with ours. Dong *et al.* [15] set a new record at 2,860 kops for point multiplication of Curve25519 on GTX 1080. Whereas, the throughput of our proposed method on Tesla P100 is 6,192 kops, 109% higher than the work in [15]. Table IX evaluates the performance of primitives for digital signature. Pan *et al.* [14] reported the existing highest throughputs



(a) Unknown PM of Curve25519 (b) Known PM of Edwards25519 (c) Unknown PM of Edwards25519

Fig. 7. Impact of skipping "simplification" on performance in Tesla P100 (the number of blocks is 56; throughput, higher is better; latency, lower is better). "Non-simplification" statistics denote the experimental results of method without skipping "simplification". "simplification-opt" statistics denote the experimental results with the optimization about skipping "simplification" in section III-E.

TABLE VII

PERFORMANCE COMPARISON WITH INTEGER-BASED IMPLEMENTATION ON GPU

Implementation	GPU	GFLOPS[*]	Primitive	PM of Curve25519 (kops)	Curve25519(scaled) (kops)	Latency (ms)
Dong et al. [15]	GTX 1080	7,967-8,873	Curve25519	2,860	2,957	7.02
Ours	Tesla P100	8,071-9,340	Curve25519	**6,192**	**6,192**	3.47

[*] The GFLOPS (Giga Floating-point operations per second) is the single-precision one. The double-precision one is 4036-4670 in Tesla P100. If the value of GFLOPS is in a range composed of two numbers, the little one is the power with the base clock and the big one is the power with the boost clock. Note that the real-time GFLOPS of GPU fluctuate within acceptable limits as the voltage and temperature change.

TABLE VIII

PERFORMANCE COMPARISON BETWEEN BERNSTEIN *et al.* [12] AND OURS

	Bernstein et al. [12]	Proposed Method
GPU	GTX 295	Tesla P100
GFLOPS	1,192.30	8,071-9,304
280-bit Curve by a 11797-bit scalar (ops)	400.70	-
Curve25519 (scaled) (Mops)	0.322	6.192
Curve448 (scaled) (Mops)	0.060	1.183

TABLE IX

PAN *et al.* [14] AND OURS

	Pan et al. [14]	Proposed Method
GPU	GTX 780i	Tesla P100
GFLOPS	5,045	8,071-9,304
NIST P-256 (Known Point) (kops)	8,710	-
Edwards25519 (Known Point) (scaled) (kops)	15,030	36,861
NIST P-256 (Unknown Point) (kops)	929	-
Edwards25519 (Unknown Point) (scaled) (kops)	1,603	3,808

of known point multiplication and unknown point multiplication of NIST P-256, i.e., 8,710/929 kops on GTX 780Ti. We scale their performance by multiplying the ratio of GFLOPS in Table IX. The throughputs of proposed known point multiplication and unknown point multiplication of Edwards25519 are still more than twice that of Pan *et al.* [14] when considering platform differences.

TABLE X

PERFORMANCE COMPARISON BETWEEN DONG *et al.* [15] AND OURS

	Dong et al. [15]	Proposed Method
Coprocessor	GTX 1080	Tesla P100
GFLOPS	7,967-8,873	8,071-9,304
PM of Curve448 (kops)	358	-
PM of Curve448 (scaled) (kops)	370	1,183

Meanwhile, the achieved latency of 0.78-3.76 ms is also within an acceptable range compared with the network traffic delay.

Curve/Edwards448 provides enough security to satisfy conservative users, but is still fast enough for those who are performance-conscious. It is useful as a more conservative supplement to Curve25519 and Edwards25519 [5]. Table X assesses the performance of Curve448 of proposed methods vs. Dong *et al.* [15]. We find that the proposed approach significantly outperforms Dong *et al.* [15] with 319% throughput. Additionally, the throughputs of known point and unknown point multiplication of Edwards448 achieve 4,143/398 kops, combining efficiency and high-level security.

This paragraph details the primary performance advantage compared with integer-based works. First, we apply larger samples to finite field arithmetic. Generally, integer-based methods use eight 32-bit samples to represent a 256-bit big number while we use larger samples as illustrated in Section IV. And fewer samples lead to fewer instructions for finite field arithmetic. The throughput of different arithmetic instructions that are natively supported in hardware brings about extra promotions. Meanwhile, we find that DPF multiplication combined with fast reductions in the integer domain is very efficient and results in significantly improved

performance over earlier works, in both the integer and floating point domains.

VI. Conclusion

In this contribution, we present the DPF-ECC framework to accelerate ECC schemes by taking full advantage of DPF computing power. Taking full advantage of DPF computing power, the DPF-ECC framework cleverly designed and exploits each bit of the DPF numbers and minimizes the overhead of data format conversion. By conducting two comprehensive case studies on Crandall primes and Solinas primes, we show that the framework is of great compatibility. In combination with DPF-ECC framework, we implement primitives of Curve25519/448 and Edwards25519/448 as examples for Crandall primes and Solinas primes respectively. In the Tesla P100, our throughput gains 2x to 3x performance improvement compared to the existing fastest work [15]. Our results demonstrate that the DPF-ECC framework is a competitive candidate for implementing ECC schemes. Although it is weak in terms of latency and the required number of concurrent requests compared with CPU or hardware based ones, higher throughput, flexibility and lower developing cost make it very promising in large-scale environments, e.g., data centers with GPUs accelerators.

Our future work will focus on the implementations of high-throughput and low-latency quantum-safe cryptography such as SIKE [51] and SIDH [52].

Appendix A
Reduction Methodology on $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$

Here gives the reduction formula of NIST P-256. Let $\mathbb{F}_p$ be a prime field where $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

As illustrated in Section III-B and Section IV-B, an element $A \in \mathbb{F}_p$ is represented by four 52-bit samples and a 48-bit sample as $(a_0, a_1, a_2, a_3, a_4)$, such that $A \equiv \sum_{i=0}^{4} a_i 2^{52i} \pmod p$, for $i \in [0, 3]$, $0 \le a_i < 2^{52}$ and $0 \le a_{l-1} < 2^{48}$.

The product C of two numbers in $\mathbb{F}_p$ is less than p^2 and can be represented as $C \equiv \sum_{i=0}^{9} c_i \cdot 2^{52i}$. Note that $2^{256} \equiv 2^{224} - 2^{192} - 2^{96} + 1 \pmod p$. Thus $C \equiv \sum_{i=0}^{4} c_i \cdot 2^{52i} + \sum_{i=5}^{9} c_i \cdot 2^{52i} \pmod p$ can be transformed into $C \equiv \sum_{i=0}^{4} c_i \cdot 2^{52i} + \sum_{i=0}^{4} 2^4 \cdot (2^{224} - 2^{192} - 2^{96} + 1) \cdot c_{i+5} \cdot 2^{52i} \pmod p$. Then reuse fast reduction method specific to the DPF-ECC framework in Section III-E until $C \pmod p$ can be reduced as $\hat{C} \equiv \sum_{i=0}^{4} \hat{c}_i \cdot 2^{52i} \pmod p$. After calculation, the terms $\hat{c}_i, i \in [0, 4]$ are given by

$$
\begin{aligned}
\hat{c}_0 &= c_0 + 2^4 c_5 + 2^{24} c_6 - 2^{12} c_7 - 2^{32} c_8 - 2^{20} c_9 \\
\hat{c}_1 &= c_1 - 2^{48} c_5 + 2^4 c_6 + 2^{24} c_7 - 2^{12} c_8 - 2^{32} c_9 - c_9 \\
\hat{c}_2 &= c_2 - 2^{16} c_6 - 2^{48} c_6 + 2^5 c_7 + 2^{25} c_8 + 2^{44} c_9 \\
\hat{c}_3 &= c_3 - 2^{40} c_5 - 2^{16} c_7 + 2^5 c_8 + 2^{25} c_9 \\
\hat{c}_4 &= c_4 + 2^{20} c_5 - 2^8 c_6 - 2^{28} c_7 - 2^{48} c_8 + 48 \cdot c9
\end{aligned}
$$

Instead of multiplying by the terms of 2^{12}, 2^{16}, 2^{20}, 2^{24}, 2^{25}, 2^{28}, 2^{38}, 2^{40}, 2^{44} and 2^{48} directly, we use the `&` and `shift` operations to prevent overflow. Let $g(x, t) = (x \& mask_t) \ll (52 - t)$ and $h(x, y, z) = (x \gg y) \ll z$ where $mask_t = 2^t - 1$. Let $r_i(z) = c_i \gg z$. The expression for $\hat{c}_i$ can be transformed into

$$
\begin{aligned}
\hat{c}_0 &= c_0 + 2^4 c_5 + g(c_6, 28) - g(c_7, 40) - g(c_8, 20) \\
&\quad - g(c_9, 32) \\
&\quad + h(c_5, 32, 4) - h(c_7, 24, 4) - h(c_8, 4, 4) - h(c_8, 36, 4) \\
\hat{c}_1 &= c_1 + 2^4 c_6 - c_9 + r_6(28) - r_7(40) - r_8(20) + g(c_7, 28) \\
&\quad - r_9(32) - g(c_5, 4) - g(c_8, 40) - g(c_9, 20) \\
&\quad + g(r_7(24), 4) \\
&\quad - g(r_5(32), 4) + g(r_8(4), 4) + g(r_8(36), 4) \\
\hat{c}_2 &= c_2 + 2^5 c_7 - r_5(4) - r_5(36) + 2r_7(28) + r_8(8) - r_9(20) \\
&\quad - g(c_6, 4) - g(c_6, 36) + g(c_8, 27) + g(c_9, 8) \\
\hat{c}_3 &= c_3 + 2^5 c_8 - r_6(36) - r_6(4) + r_8(27) + r_9(8) - g(c_5, 12) \\
&\quad - g(r_5(32), 12) - g(c_7, 36) + g(r_7(24), 12) \\
&\quad + g(r_8(36), 12) \\
&\quad + g(r_8(4), 12) + g(c_9, 27) \\
\hat{c}_4 &= c_4 + 48 c_9 - 2^8\ c_6 - r_5(12) - r_5(44) + r_8(16) + r_9(27) \\
&\quad + r_8(48) - g(c_7, 24) - h(c_7, 24, 20) + g(c_5, 32) \\
&\quad - g(c_8, 4) \\
&\quad + g(r_5(32), 20) - g(r_8(4), 32) - h(c_8, 36, 20).
\end{aligned}
$$

Appendix B
Reduction Methodology on $p = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$

Here gives the reduction formula of NIST P-384. Let $\mathbb{F}_p$ be a prime field where $p = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$.

As illustrated in Section III-B and Section IV-B, an element $A \in \mathbb{F}_p$ is represented by eight 48-bit samples as $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$, such that $A \equiv \sum_{i=0}^{7} a_i 2^{48i} \pmod p$, for $i \in [0, 7]$, $0 \le a_i < 2^{48}$.

The product C of two numbers in $\mathbb{F}_p$ is less than p^2 and can be represented as $C = \sum_{i=0}^{15} c_i \cdot 2^{48i}$. Note that $2^{384} \equiv 2^{128} + 2^{96} - 2^{32} + 1 \pmod p$. Thus $C \equiv \sum_{i=0}^{7} c_i \cdot 2^{48i} + \sum_{i=8}^{15} c_i \cdot 2^{48i} \pmod p$ can be transformed into $C \equiv \sum_{i=0}^{7} c_i \cdot 2^{48i} + \sum_{i=0}^{7} (2^{128} + 2^{96} + 2^{32} - 1) \cdot c_{i+8} \cdot 2^{48i} \pmod p$. Then reuse fast reduction method specific to the DPF-ECC framework in Section III-E until $C \pmod p$ can be reduced as

$$
\begin{aligned}
C \equiv & \sum_{i=0}^{7} (c_i + c_{i+8} - 2^{32} \cdot c_{i+8}) \cdot 2^{48i} + \sum_{i=0}^{1} c_{i+14} \cdot 2^{48i} \\
& + \sum_{i=2}^{7} (2^{32} + 1) \cdot c_{i+6} \cdot 2^{48i} - \sum_{i=1}^{2} 2^{16} \cdot c_{i+13} \cdot 2^{48i} \\
& + \sum_{i=2}^{3} (2^{33} + 1) \cdot c_{i+12} \cdot 2^{48i} \\
& + \sum_{i=3}^{4} 2^{16} \cdot c_{i+11} \cdot 2^{48i} (mod\ p).
\end{aligned}
$$

Instead of multiplying by the terms of 2^{16}, 2^{32} and 2^{33} directly, we use the `&` and `shift` operations to prevent overflow. Let $g(x, t) = (x \& mask_t) \ll (48 - t)$ where

$mask_t = 2^t - 1$. Let $r_i(z) = c_i \gg z$. The expression for $C \bmod p$ can be transmitted into

$$\begin{aligned}
C \equiv & \sum_{i=0}^{7}(c_i + c_{i+8}) \cdot 2^{48i} + \sum_{i=2}^{7} c_{i+6} \cdot 2^{48i} + \sum_{i=0}^{1} c_{i+14} \cdot 2^{48i} \\
& - \sum_{i=0}^{7} g(c_{i+8}, 16) \cdot 2^{48i} - \sum_{i=1}^{7} r_{i+7}(16) \cdot 2^{48i} + \sum_{i=2}^{2}[c_{i+11} \\
& - g(r_{i+13}(16), 16) - r_{i+13}(16) + g(r_{i+11}(16), 16)] \cdot 2^{48i} \\
& + \sum_{i=3}^{3}[r_{i+10}(32) - r_{i+12}(32)] \cdot 2^{48i} + \sum_{i=0}^{0}[r_{i+13}(16) \\
& + g(r_{i+15}(16), 16) - g(r_{i+13}(16), 16) - r_{i+15}(16)] \cdot 2^{48i} \\
& + \sum_{i=1}^{1}[r_{i+14}(32) - r_{i+12}(32)] \cdot 2^{48i} + \sum_{i=2}^{7} g(c_{i+6}, 16) \cdot 2^{48i} \\
& + \sum_{i=3}^{4}[r_{i+11}(15) + g(c_{i+11}, 32)] \cdot 2^{48i} + \sum_{i=4}^{5} r_{i+10}(32) \cdot 2^{48i} \\
& - \sum_{i=1}^{2} g(c_{i+13}, 32) \cdot 2^{48i} + \sum_{i=3}^{7} r_{i+5}(16) \cdot 2^{48i} \\
& + \sum_{i=2}^{3}[g(c_{i+12}, 15) - r_{i+12}(32) + c_{i+12}] \cdot 2^{48i} (mod\ p)
\end{aligned}$$

DISCLOSURE

A preliminary version of this paper appeared under the title *DPF-ECC: Accelerating Elliptic Curve Cryptography with Floating-Point Computing Power of GPUs,* in Proc. 2020 IEEE 34th International Parallel and Distributed Processing Symposium, New Orleans, Louisiana, May 18–22, 2020 [53].

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for useful comments.

REFERENCES

[1] V. S. Miller, "Use of elliptic curves in cryptography," in *Proc. Conf. Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1985, pp. 417–426.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, no. 177, pp. 203–209, 1987.

[3] U.S. Department of Commerce, National Institute of Standards and Technology. (2013). *Digital Signature Standard (DSS)*. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[4] A. Langley, M. Hamburg, and S. Turner, *Elliptic Curves for Security*, document RFC 7748, Jan. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7748.txt

[5] S. Josefsson and I. Liusvaara, *Edwards-Curve Digital Signature Algorithm (EdDSA)*, document RFC 8032, Jan. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8032.txt

[6] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, document RFC 8446, Aug. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8446.txt

[7] A. Adamantiadis, S. Josefsson, and M. D. Baushke, *Secure Shell (SSH) Key Exchange Method Using Curve25519 and Curve448*, document RFC 8731, 2020, pp. 1–6, doi: 10.17487/RFC8731.

[8] B. Harris and L. Velvindron, *Ed25519 and Ed448 Public Key Algorithms for the Secure Shell (SSH) Protocol*, document RFC 8709, 2020, pp. 1–7, doi: 10.17487/RFC8709.

[9] F. Hao, *J-Pake: Password-Authenticated Key Exchange by Juggling*, document RFC 8236, 2017.

[10] C. Briefing. (2020). *China's Double 11 Shopping Festival Tests Consumption Strength After COVID-19*. [Online]. Available: https://www.china-briefing.com/news/chinas-double-11-shopping-festival-tests-consumption-strength-after-covid-19/

[11] R. Szerwinski and T. Güneysu, "Exploiting the power of GPUs for asymmetric cryptography," in *Cryptographic Hardware and Embedded Systems—CHES 2008*. Berlin, Germany: Springer, 2008, pp. 79–99.

[12] D. J. Bernstein, T.-R. Chen, C.-M. Cheng, T. Lange, and B.-Y. Yang, "ECM on graphics cards," in *Advances in Cryptology—EUROCRYPT 2009*. Berlin, Germany: Springer, 2009, pp. 483–501.

[13] F. Zheng, W. Pan, J. Lin, J. Jing, and Y. Zhao, "Exploiting the potential of GPUs for modular multiplication in ECC," in *Proc. 15th Int. Workshop Inf. Secur. Appl. (WISA)*, Jeju Island, South Korea, Aug. 2014, pp. 295–306.

[14] W. Pan, F. Zheng, Y. Zhao, W.-T. Zhu, and J. Jing, "An efficient elliptic curve cryptography signature server with GPU acceleration," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 1, pp. 111–122, Jan. 2017.

[15] J. Dong, F. Zheng, J. Cheng, J. Lin, W. Pan, and Z. Wang, "Towards high-performance X25519/448 key agreement in general purpose GPUs," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, May 2018, pp. 1–9.

[16] F. Zheng, W. Pan, J. Lin, J. Jing, and Y. Zhao, "Exploiting the floating-point computing power of GPUs for RSA," in *Proc. 17th Int. Conf. Inf. Secur. (ISC)*, Hong Kong, Oct. 2014, pp. 198–215.

[17] J. Dong, F. Zheng, W. Pan, J. Lin, J. Jing, and Y. Zhao, "Utilizing the double-precision floating-point computing power of GPUs for RSA acceleration," *Secur. Commun. Netw.*, vol. 2017, pp. 1–15, Sep. 2017.

[18] N. Emmart, F. Zheng, and C. Weems, "Faster modular exponentiation using double precision floating point arithmetic on the GPU," in *Proc. IEEE 25th Symp. Comput. Arithmetic (ARITH)*, Jun. 2018, pp. 130–137.

[19] J. Dong, F. Zheng, N. Emmart, J. Lin, and C. Weems, "SDPF-RSA: Utilizing floating-point computing power of GPUs for massive digital signature computations," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2018, pp. 599–609.

[20] S. Akleylek and Z. Y. Tok, "Efficient arithmetic for lattice-based cryptography on GPU using the CUDA platform," in *Proc. 22nd Signal Process. Commun. Appl. Conf. (SIU)*, Apr. 2014, pp. 854–857.

[21] N. Gupta, A. Jati, A. K. Chauhan, and A. Chattopadhyay, "PQC acceleration using GPUs: FrodoKEM, NewHope, and kyber," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 575–586, Mar. 2021.

[22] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Accelerating fully homomorphic encryption using GPU," in *Proc. IEEE Conf. High Perform. Extreme Comput.*, Sep. 2012, pp. 1–5.

[23] W. Dai and B. Sunar, "cuHE: A homomorphic encryption accelerator library," in *Proc. Int. Conf. Cryptogr. Inf. Secur. Balkans*. Cham, Switzerland: Springer, 2015, pp. 169–186.

[24] A. Al Badawi, B. Veeravalli, C. F. Mun, and K. M. M. Aung, "High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA," in *Proc. Trans. Cryptograph. Hardw. Embedded Syst. (IACR)*, 2018, pp. 70–95.

[25] National Institute of Standards and Technology. (2016). *HSM-ZJ2014 FIPS 140-2 Non-Proprietary Security Policy*. [Online]. Available: https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/2692

[26] D. J. Bernstein *et al.*, "The billion-mulmod-per-second PC," in *Proc. Workshop Rec. SHARCS*, vol. 9, 2009, pp. 131–144.

[27] E. B. Barker, D. Johnson, and M. E. Smid, *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)*, document SP 800-56A, 2007.

[28] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Math. Comput.*, vol. 48, no. 177, pp. 243–264, Jan. 1987. [Online]. Available: http://www.jstor.org/stable/2007888

[29] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Advances in Cryptology—ASIACRYPT 2007*, K. Kurosawa, Ed. Berlin, Germany: Springer, 2007, pp. 29–50.

[30] T. L. Daniel and J. Bernstein. (2018). *SafeCurves: Choosing Safe Curves for Elliptic-Curve Cryptography*. [Online]. Available: https://safecurves.cr.yp.to/

[31] C. Research. (2010). *SEC 2: Recommonded Elliptic Curve Domain Parameters*. [Online]. Available: http://www.secg.org/sec2-v2.pdf

[32] E. Brainpool. (Oct. 2005). *ECC Brainpool Standard Curves and Curve Generation*. [Online]. Available: http://www.ecc-brainpool.org/download/Domain-parameters.pdf

[33] National Security Agency. (2015). *NSA Suite B Cryptography*. [Online]. Available: https://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml

[34] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comput. Surveys*, vol. 23, no. 1, pp. 5–48, Mar. 1991.
[35] T. J. Dekker, "A floating-point technique for extending the available precision," *Numer. Math.*, vol. 18, no. 3, pp. 224–242, 1971.
[36] R. Crandall, "Method and apparatus for public key exchange in a cryptographic system," US Patent 5 159 632, Oct. 27, 1992.
[37] J. A. Solinas, *Generalized Mersenne Numbers*. Waterloo, ON, Canada: Univ. of Waterloo, 1999.
[38] J. Solinas, *Generalized Mersenne Prime*. Boston, MA, USA: Springer, 2005, pp. 239–240, doi: 10.1007/0-387-23483-7_174
[39] D. F. Aranha, P. S. L. M. Barreto, G. C. C. F. Pereira, and J. E. Ricardini, "A note on high-security general-purpose elliptic curves," Cryptol. ePrint Arch., UCSD, La Jolla, CA, USA, Tech. Rep. 2013/647, 2013. [Online]. Available: https://eprint.iacr.org/2013/647
[40] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange, "Elligator: Elliptic-curve points indistinguishable from uniform random strings," Cryptol. ePrint Arch., Berlin, Germany, Tech. Rep. 2013/325, 2013. [Online]. Available: https://eprint.iacr.org/2013/325
[41] D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *Public Key Cryptography—PKC 2006*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Berlin, Germany: Springer, 2006, pp. 207–228.
[42] T. Lange and D. J. Bernstein. (Sep. 2013). *Security Dangers of the NIST Curves*. [Online]. Available: https://cr.yp.to/talks/2013.09.16/slides-djb-20130916-a4.pdf
[43] M. Hamburg, "Ed448-goldilocks, a new elliptic curve," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 625, Jun. 2015.
[44] C. K. Koç, "Analysis of sliding window techniques for exponentiation," *Comput. Math. Appl.*, vol. 30, no. 10, pp. 17–24, 1995.
[45] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2007, pp. 29–50.
[46] S. Antao, J.-C. Bajard, and L. Sousa, "Elliptic curve point multiplication on GPUs," in *Proc. 21st IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2010, pp. 192–199.
[47] S. Antao, J.-C. Bajard, and L. Sousa, "RNS-based elliptic curve point multiplication for massive parallel architectures," *Comput. J.*, vol. 55, no. 5, pp. 629–647, May 2012.
[48] S. Pu and J.-C. Liu, "EAGL: An elliptic curve arithmetic GPU-based library for bilinear pairing," in *Pairing-Based Cryptography–Pairing 2013*. Cham, Switzerland: Springer, 2014, pp. 1–19.
[49] J. W. Bos, "Low-latency elliptic curve scalar multiplication," *Int. J. Parallel Program.*, vol. 40, no. 5, pp. 532–550, Oct. 2012.
[50] E. Mahe and J.-M. Chauvet, "Fast GPGPU-based elliptic curve scalar multiplication," *IACR Cryptol. ePrint Arch.*, vol. 2014, p. 198, Sep. 2014.
[51] D. Jao *et al.*, "SIKE: Supersingular isogeny key encapsulation," in *NIST Standardization Process On Post-Quantum Cryptography*. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2017.
[52] D. Jao and L. D. Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," in *Proc. Post-Quantum Cryptogr.-Int. Workshop*, 2011, pp. 19–34.
[53] L. Gao, F. Zheng, N. Emmart, J. Dong, J. Lin, and C. Weems, "DPF-ECC: Accelerating elliptic curve cryptography with floating-point computing power of GPUs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2020, pp. 494–504.

Lili Gao received the B.E. degree from Anhui University. She is currently pursuing the Ph.D. degree with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences. Her research interests include applied cryptography and high-performance computing.

Fangyu Zheng received the B.E. degree from the University of Science and Technology of China in 2011 and the Ph.D. degree from the University of Chinese Academy of Sciences in 2016. He is currently an Assistant Professor with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences. His research interests include applied cryptography and high-performance computing.

Rong Wei received the B.E. degree from the Beijing Institute of Technology. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences. His research interest includes applied cryptography.

Jiankuo Dong received the B.E. degree from Xi'an Jiaotong University in 2014 and the Ph.D. degree from the University of Chinese Academy of Sciences in 2019. He is currently an Assistant Professor with the School of Computer Science, Nanjing University of Posts and Telecommunications. His research interests include cryptographic engineering and public key cryptography.

Niall Emmart received the B.S. degree in pure mathematics, the M.S. degree in computer science, and the Ph.D. degree in computer science from the University of Massachusetts Amherst, Amherst, MA, USA, in 1992, 2013, and 2018, respectively.

Yuan Ma received the B.E. degree from Tsinghua University in 2009, and the Ph.D. degree from the University of Chinese Academy of Sciences in 2014. He is currently an Associate Professor with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, and the Data Assurance and Communication Security Research Center, Chinese Academy of Sciences. His research interests include physical random number generator design and evaluation, cryptographic algorithm high-speed implementation, and hardware security module.

Jingqiang Lin (Senior Member, IEEE) received the M.S. and Ph.D. degrees from the University of Chinese Academy of Sciences in 2004 and 2009, respectively. He is currently a Full Professor with the School of Cyber Security, University of Science and Technology of China. His research interests include applied cryptography and system security.

Charles Weems (Senior Member, IEEE) received the B.S. degree (Hons.) and the M.A. degree from Oregon State University in 1977 and 1979, respectively, the M.S. degree in computer science and the Ph.D. from the University of Massachusetts, Amherst in 1984 and 2013, respectively. He is the Co-Director of the Architecture and Language Implementation Laboratory, University of Massachusetts. His current research interests include GPU computing and high-precision arithmetic. He is a Distinguished Member of ACM and a member of the Executive Committee of the IEEE TC on Parallel Processing. He was a recipient of the IEEE Taylor Booth Award and the University Distinguished Teaching Award.