Speeding up Elliptic Curve Cryptography on the P-384 Curve

Armando Faz-Hernández, Julio López*

Institute of Computing, University of Campinas.
1251 Albert Einstein, Cidade Universitária. Campinas, Brazil.

{armfazh,jlopez}@ic.unicamp.br

Abstract. *The P-384 is one of the standardized elliptic curves by ANSI and NIST. This curve provides a 192-bit security level and is used in the computation of digital signatures and key-agreement protocols. Although several publicly-available cryptographic libraries support the P-384 curve, they have a poor performance. In this work, we present software techniques for accelerating cryptographic operations using the P-384 curve; first, we use the latest vector instructions of Intel processors to implement the prime field arithmetic; second, we devise a parallel scheduling of the complete formulas for point addition law. As a result, on Skylake micro-architecture, our software implementation is 15% and 40% faster than the OpenSSL library for computing ECDSA signatures and the ECDH protocol, respectively.*

Resumo. *A P-384 é uma das curvas elípticas padronizadas pelo ANSI e o NIST. Ela fornece um nível de segurança de 192 bits e é usada tanto na computação de assinaturas digitais como nos protocolos de acordo de chaves. Embora várias bibliotecas criptográficas disponíveis publicamente suportam a P-384, elas possuem um baixo desempenho. Neste trabalho, apresentamos técnicas de implementação em* software *para acelerar operações criptográficas usando a curva P-384; primeiro, usamos as mais novas instruções vetoriais dos processadores Intel para implementar a aritmética de corpo primo; depois, propomos um escalonamento paralelo das fórmulas completas para calcular a lei de adição de pontos. Como resultado, na microarquitetura* Skylake*, a nossa implementação em* software *é 15% e 40% mais rápida do que a biblioteca OpenSSL para calcular assinaturas ECDSA e o protocolo ECDH, respectivamente.*

1. Introduction

The elliptic curve cryptography (ECC) is well-known for providing secure algorithms that are faster and use shorter key sizes in comparison with practical cryptosystems like the RSA [Rivest et al. 1978]. In July 1999, the National Institute of Standards and Technology (NIST) recommended a set of elliptic curves and parameters to be used by federal government applications [NIST 1999]. This recommendation considers a set of 15 elliptic curves that covers five security levels: 80, 112, 128, 192, and 256 bits. Five of these curves are defined over prime fields and are known as P-192, P-224, P-256, P-384, and P-521. Afterward, the American National Standards Institute (ANSI) standardized the use of NIST's curves [ANSI 1999]; and this had an impact on the widespread use of ECC for establishing secure communications. For instance, a scan performed in 2014 over around

*The authors gratefully acknowledge the support from the Intel Labs University Research Office.

 ©2016 SBC — Soc. Bras. de Computação

Table 1. Set of algorithms classified according to the security levels defined by Suite B. Entries marked with ⋆ denote algorithms considered for legacy support.

Primitive	Algorithm	Suite B Security Level	
		SECRET	TOP SECRET
Data Encryption	AES	AES-128	AES-256
Hash Function	SHA2	SHA-256	SHA-384
Key Agreement	ECDH	P-256	P-384
	DH⋆	2,048-bit modulus	3,072-bit modulus
Digital Signature	ECDSA	P-256	P-384
	DSA⋆	2,048-bit modulus	3,072-bit modulus
	RSA⋆	2,048-bit modulus	3,072-bit modulus

30.2 millions of Internet hosts revealed that 7.2% of them support a cipher suite based on elliptic curves [Bos et al. 2014]. The authors also showed that 98% of these hosts support the P-256 curve, 80% support the P-384 curve, and only 17% support the P-521 curve. The use and application of ECC are nowadays a suitable alternative for implementing public-key cryptography systems.

The Committee on National Security Systems (CNSS) in the United States selected a set of standardized security protocols and cryptographic algorithms to define a cipher suite called *Suite B* [CNSS 2012]. The Suite B defines two security levels: the `SECRET` level and the `TOP SECRET` level, detailed in Table 1. The most remarkable of Suite B is the choice of elliptic curves as the preferred cryptosystem over other public-key algorithms such as RSA and DSA. Recently in 2015, the CNSS announced an update concerning the usage of the Suite B [CNSS 2015]. The announcement was motivated by the effect of quantum computing on information assurance; in particular, the weakness of ECC against an attack by a quantum computer [Proos and Zalka 2003]. At this time, the latest quantum computers do not represent a threat to the ECC instances approved in standards. Thus, before making a transition to quantum-resistant algorithms, the CNSS restricted the use of the Suite B to work only at the `TOP SECRET` level.

In some computational systems, like those dependent on a cryptographic hardware infrastructure, upgrading to the `TOP SECRET` level could incur into a serious investment for updating the whole infrastructure in cases where the higher security level is not supported. However, in systems using a cryptographic software infrastructure, this upgrade can be performed by setting the appropriate parameters in the software libraries. Unfortunately, a loss of performance can occur, since some cryptographic libraries have inefficient implementations for the higher security levels, like in the case of the P-384 curve. Consequently, the transition to the `TOP SECRET` level enhances security but downgrades on performance.

Regarding software implementations, the increasing support for parallel computing in the latest micro-architectures favors algorithms that can be partitioned into a series of independent tasks. Thus, it is necessary to provide new algorithms, or adapt the existent ones, to take advantage of modern processors. In that sense, it is crucial to devise formulations that increase the parallelism degree of algorithms that support ECC.

 ©2016 SBC — Soc. Bras. de Computação

In this work, we focus on a high-performance software implementation of the P-384 curve using advanced software techniques and efficient algorithms for the secure execution of cryptographic operations. The following is the synopsis of our contributions. In a lower level, we optimize the execution of prime field arithmetic by using vector instruction sets. At a higher level, we devise a scheduling of the field operations employed in the *complete* formulas for calculating point additions; this scheduling allowed that point additions be executed in parallel with low communication between the processing units. To determine the impact of the techniques proposed, we apply them to the implementation of the Elliptic Curve Diffie-Hellman protocol (ECDH) [ANSI 2001] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [Vanstone et al. 1992]. Finally, we compare the performance of our implementation against some publicly-available cryptographic libraries.

Now, we detail some software optimizations targeting the NIST's curves. Käsper presented a fast implementation of the P-224 curve using a redundant representation for implementing the prime field arithmetic [Käsper 2012]. Gueron and Krasnov showed optimizations for the P-256 curve by computing the field multiplication efficiently using Montgomery-friendly primes [Gueron and Krasnov 2014]. Granger and Scott described a novel technique for accelerating multiplications in the field $\mathbb{F}_{2^{521}-1}$, which they used for the implementation of the P-521 curve; this technique can also be extended to Crandall's numbers [Granger and Scott 2015]. Unfortunately, from these prime field arithmetic optimizations, only the redundant representation applies to the prime modulus used by the P-384 curve.

2. Elliptic Curve Cryptography

An elliptic curve in short Weierstrass form over $\mathbb{F}_p$ is defined by $E_{a,b}(\mathbb{F}_p)\colon y^2 = x^3 + ax + b$, where $a, b \in \mathbb{F}_p$ and $-16(4a^3 + 27b^2) \neq 0$. The set of points belonging to this curve forms a commutative group E of cardinality ℓ where $\mathcal{O}$ is the identity element. The group law is denoted by $+$; thus, for any $P, Q \in E$, the operation $P + Q$ is called as *point addition*; in the case that $P = Q$, then the operation $2P = P + P$ is called as *point doubling*. The inverse of a point $P = (x, y)$ is calculated as $-P = (x, -y)$.

One of the most critical operations in ECC is the *point multiplication*. Given a point $P \in E$ and an integer k, the point multiplication is defined as the scalar product $kP = P + P + \cdots + P$, i.e. the point obtained after P was added to itself k times. The most efficient algorithms to calculate point multiplication have a computational complexity of $O(\log k)$; refer to [Hankerson et al. 2003] for a complete reference about point multiplication algorithms.

Around 1986, Miller [Miller 1986] and Koblitz [Koblitz 1987] suggested the use of elliptic curves for constructing public-key cryptosystems under the assumption that the *elliptic curve discrete logarithm problem* (ECDLP) is computationally intractable. This problem is stated as follows: given a generator point G of E and a point $P \in E$, the ECDLP consists on finding a number $k \in \mathbb{Z}_\ell$ such that $P = kG$. The best-known algorithm to solve ECDLP has a computational complexity of $O(\sqrt{\ell})$ [Pollard 1975]. Hence, key sizes on ECC are shorter than on RSA for attaining the same security level.

The elliptic curve Diffie-Hellman protocol (ECDH) is used to establish a shared secret between two entities securely. Assume that Alice and Bob want to agree on a

 ©2016 SBC — Soc. Bras. de Computação

Domain parameter generation

Given: m, the security parameter.
Return: $D = \{p, E, G, \ell, h\}$, where p is a prime number such that $p \approx 2^{2m}$, E is the group induced by $E_{a,b}(\mathbb{F}_p)$, $G \in E$ is a generator point of order ℓ, and h is a hash function producing $2m$ bits.

Key Generation

Given: The domain parameters D.

1. Choose $\mathbf{sk} \xleftarrow{\$} \mathbb{Z}_\ell^*$.
2. Calculate $\mathbf{pk} \leftarrow \mathbf{sk}G$.

Return: $(\mathbf{sk}, \mathbf{pk})$.

Signature generation

Given: The domain parameters D, a message M and a secret key $\mathbf{sk}$.

1. Choose $k \xleftarrow{\$} \mathbb{Z}_\ell^*$.
2. Calculate $(R_x, R_y) \leftarrow kG$.
3. Calculate $r \leftarrow R_x \bmod \ell$.
4. If $r = 0$ then go to Step 1.
5. Set $s \leftarrow k^{-1}(h(M)+\mathbf{sk}\times r) \bmod \ell$.
6. If $s = 0$ then go to Step 1.

Return: (r, s).

Verification

Given: The domain parameters D, a message M, a signature (r, s) and a public key $\mathbf{pk}$.

1. Set $u_1 \leftarrow s^{-1} \times h(M) \bmod \ell$.
2. Set $u_2 \leftarrow s^{-1} \times r \bmod \ell$.
3. Calculate $(x, y) \leftarrow u_1G + u_2\mathbf{pk}$.
4. If $x \equiv r \bmod \ell$, set $v \leftarrow$ Accept; otherwise, set $v \leftarrow$ Reject.

Return: v.

Figure 1. Set of algorithms for the ECDSA scheme.

shared secret S. Then, both Alice and Bob create a pair of keys, $(\mathrm{s}_i, \mathrm{P}_i)$, where the secret key is randomly chosen as $\mathrm{s}_i \xleftarrow{\$} \mathbb{Z}_\ell^*$, and the public key is calculated as $\mathrm{P}_i \leftarrow \mathrm{s}_iG$ for $i \in \{A, B\}$. After that, they communicate their public keys to each other. To calculate the shared secret S, both multiply their private key by the public key received; thus, $S = \mathrm{s}_A\mathrm{P}_B = \mathrm{s}_B\mathrm{P}_A$. The main application of this protocol is for generating secret keys for symmetric data encryption. The ECDH protocol was standardized by ANSI [ANSI 2001] and by NIST [Barker et al. 2007].

The elliptic curve digital signature algorithm (ECDSA) is an adaptation of the digital signature algorithm (DSA) [NIST 2000] that replaced the use of the group $\mathbb{Z}_q$ by the elliptic curve group E. Figure 1 shows the set of algorithms of the ECDSA. The ANSI standardized the ECDSA [ANSI 1999]; this had a strong influence on other agencies, such as IEEE [IEEE 2000] and SECG [Brown 2009] that also approved its use.

3. Prime Field Arithmetic

3.1. Targeting the Machine Instruction Set

The latest processors strive to boost the performance of applications by improving on both the instruction-level parallelism (ILP) and the data-level parallelism (DLP). Widening the execution engine of a processor enhances the ILP; this means that by increasing the number of functional units will allow that several units execute instructions. For increasing DLP, some micro-architectures have included a vector unit able to execute one instruction over a set of words; this unit operates in concordance with the parallel paradigm known as Single Instruction Multiple Data (SIMD). We focus on the use of vector instructions to increase the performance of field arithmetic operations.

 ©2016 SBC — Soc. Bras. de Computação

In 1999, the MMX was the first vector instruction set integrated into the Intel processors; the MMX supported operations over vector registers of 64 bits. Since then, the vector unit evolved into a more complex unit supporting both 128- and 256-bit vector instruction sets; on Intel processors these instruction sets are known as SSE, AVX and AVX2 [Intel Corporation 2011]. The AVX2 unit has 16 vector registers of 256 bits and also provides instructions for computing operations over words of 64 bits. The first micro-architecture supporting AVX2 was codenamed as *Haswell* and was released in 2014. Nowadays, both the *Broadwell* and the *Skylake* micro-architectures also have support for 256-bit vector instructions. All these micro-architectures contain two functional units for integer arithmetic instructions, three units for logic instructions, one unit for shift and multiplication instructions (two units in the case of Skylake) and one unit for byte permutations. A complete list of AVX2 instructions is available at [Intel Corporation 2011].

The most critical instruction in the implementation of prime field arithmetic is the integer multiplier. In AVX2, the `VPMULUDQ` instruction computes four simultaneous multiplications of 32-bit words producing four 64-bit products. This instruction takes five clock cycles; meanwhile, the 64-bit native multiplier, provided by the instructions `MULQ` and `MULX`, takes three clock cycles. This fact presents a trade-off between parallel computation and latency.

3.2. Field Element Representation Suitable for Vector Instructions

The prime modulus used by P-384 is $p_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$; this number belongs to the Generalized Mersenne numbers [Solinas 1999] which were designed to perform the modular reduction efficiently.

The implementation of operations in $\mathbb{F}_{p_{384}}$ requires operating over integers of 384 bits. Usually, field elements are split in digits, having a size equal to word size of the machine. One downside of this method appears on the implementation of modular additions, which propagate carry bits from the first to the last digit; this propagation introduces a dependency chain on computations. It has been observed that large dependency chains reduce the throughput of a calculation significantly. As a corollary, it follows that reducing such dependencies is crucial to improve the performance.

In that sense, we express field elements using a *redundant* representation with the aim of reducing carry dependencies [Käsper 2012]. In this representation, a field element is split into digits of smaller size that the word size of the instruction set; for example, an element of $\mathbb{F}_{p_{384}}$ can be represented by 14 digits of 28 bits using a 32-bit instruction set or by seven digits of 55 bits using a 64-bit instruction set. This representation guarantees that the addition of two elements will not overflow the registers, in that case, the carry-bit propagation can be postponed. Moreover, a determined number of additions can be processed without any carry propagation, which enables a parallel processing of digits and leading to an immediate application of vector instructions.

We represent a field element using 14 digits of 28 bits. Operating with digit size less than 32 is justified due to the `VPMULUDQ` instruction multiplies words of 32 bits producing 64-bit products; these products can be operated using 64-bit integer arithmetic, which is supported by AVX2. We considered to use 13 digits of 30 bits; however, digits will not have enough room to store carry bits. The same argument applies to digits of 29 bits. In the other hand, using less than 28 bits per digit increases the number of digits and

 ©2016 SBC — Soc. Bras. de Computação

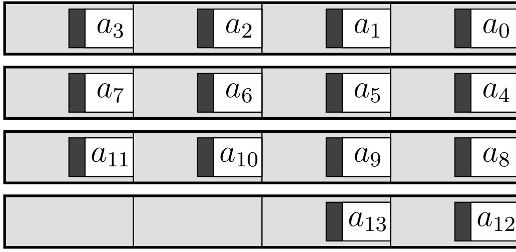

Figure 2. We store the digits a_i into four 256-bit vector registers; each vector register is composed of four 64-bit words. For computing additions, digits will have enough room (dark shading space) for storing carry bits.

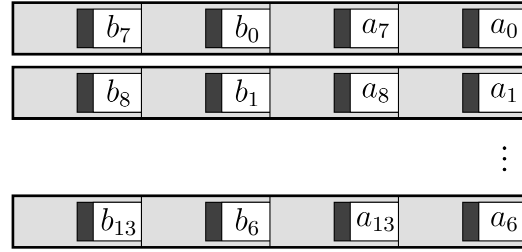

Figure 3. We store the digits of A and B into seven 256-bit vector registers. This arrangement is intended to reduce the use of permutation instructions since they move words across the 128-bit boundary increasing the execution latency.

slows field operations.

An element $a \in \mathbb{F}_{p_{384}}$ is then split into 14 digits a_i such that the following relation holds $a \equiv \sum_{i=0}^{13} 2^{28i} a_i \bmod p_{384}$. We denote by A the ordered sequence of digits $(a_0, \ldots, a_{13})$, for which a digit d in the n-th position of the sequence represents the number $2^{28n}d$. Figure 2 shows how we store one sequence of digits into four 256-bit registers.

3.3. Implementation of Arithmetic Operations

The addition $c = a+b \in \mathbb{F}_{p_{384}}$ is computed entirely in parallel; we proceed by performing $c_i = a_i + b_i$ for all $i \in [0, 14)$; using the representation shown in Figure 2 there are required only four vector addition instructions (VPADDQ). For the case of the subtraction $d = a - b \in \mathbb{F}_{p_{384}}$, we restrict digits to be always positive; hence, we compute $d_i = a_i - b_i + 2p_i$ for all $i \in [0, 14)$, where p_i is the redundant representation of p_{384}.

Now, we show how to compute the multiplication $a \times b \in \mathbb{F}_{p_{384}}$. Our technique is different from other implementations that process first the integer multiplication and then the modular reduction; instead, we perform modular reduction during the processing of integer multiplication. We express $c = a \times b$ in terms of an auxiliary function π as follows $c = \sum_{i=0}^{13} \pi^i(a) b_i$; where π is defined as $\pi(x) = 2^{28}x \bmod p_{384}$. Algorithm 1 shows the processing of modular multiplication using π. Lines 1 and 4 of Algorithm 1 calculate component-wise additions and multiplications over sequences of 14 digits; therefore, we can schedule 14 operations without dependency between them. This fact is crucial for improving performance; particularly, the processor can issue VPMULUDQ instructions into the pipeline every clock cycle since products do not present any dependency that could stall the pipeline.

The π function, used in line 3 of Algorithm 1, is computed by Algorithm 2 as follows. Given an input sequence A, every digit of A must be multiplied by 2^{28}, this can be done by changing a digit from the n-th position to the $(n+1)$-th position in the output sequence (line 1 of Algorithm 2). Now, notice that the a'_{14} represents the number $L = 2^{392}a_{13}$, which is congruent to $r = (2^{136} + 2^{104} - 2^{40} + 2^8)a_{13} \bmod p_{384}$. Thus, we add r to the digits of the output sequence; this addition must be processed ensuring that the digits do not be greater than 2^{32} since $\pi(A)$ will be the input of another round of

Algorithm 1 Modular multiplication in terms of the π function.

Input: $A = (a_0, \ldots, a_{13})$ and $B = (b_0, \ldots, b_{13})$, be sequences of the digits of a and b, respectively.

Output: $C = (c_0, \ldots, c_{13})$, be a sequence of digits such that

$$a \times b \equiv \sum_{i=0}^{13} 2^{28i} c_i \bmod p_{384}.$$

1: $C \leftarrow A \cdot b_0$
2: **for** $i \leftarrow 1$ **to** 13 **do**
3: $\quad A \leftarrow \pi(A)$ *(Alg. 2.)*
4: $\quad C \leftarrow C + A \cdot b_i$
5: **end for**
6: **return** C

Algorithm 2 The π function.

Input: $A = (a_0, \ldots, a_{13})$, be a sequence of the digits of a.

Output: $(a'_0, \ldots, a'_{13})$, be a sequence of digits such that $2^{28}a \equiv \sum_{i=0}^{13} 2^{28i} a'_i \bmod p_{384}$.

1: $a'_{i+1} \leftarrow a_i$ for all $i \in [0, 14)$
2: $L \leftarrow a'_{14}$
3: $a'_0 \leftarrow 2^8(L \bmod 2^{20})$
4: $a'_1 \leftarrow a'_1 + \lfloor L/2^{20} \rfloor - 2^{12}(L \bmod 2^{16})$
5: $a'_2 \leftarrow a'_2 - \lfloor L/2^{16} \rfloor$
6: $a'_3 \leftarrow a'_3 + 2^{20}(L \bmod 2^8)$
7: $a'_4 \leftarrow a'_4 + \lfloor L/2^8 \rfloor + 2^{24}(L \bmod 2^4)$
8: $a'_5 \leftarrow a'_5 + \lfloor L/2^4 \rfloor$
9: **return** $(a'_0, \ldots, a'_{13})$

multiplications. We guarantee this property by adding the correspondent portions of bits of a_{13} into the digits of the output sequence as it is described in lines 2-8 of Algorithm 2. The operations of Algorithm 2 are implemented using only logic and shift instructions; moreover, whenever the shift displacement is multiple of 8, we use byte permutations instead of shifts since permutations are executed by another functional unit. Therefore, by taking advantage of the out-of-order execution, instructions for π are executed while the long-latency multiplication instructions are processed increasing the ILP of this workload.

4. Elliptic Curve Arithmetic

The elliptic curve addition law is calculated using operations in the prime field such as additions, multiplications, and squares that we denote by **A**, **M**, and **S**, respectively. Adding points in affine coordinates implies the use of field inversions, which in general are computationally expensive.

An alternative to avoid inversions consists on representing points using Jacobian coordinates, i.e. a point $P = (x, y)$ is represented by (X, Y, Z) such that $x = X/Z^2$ and $y = Y/Z^3$. This coordinate system presents the most efficient formulas to compute the addition law for curves in the short Weierstrass form. Specifically, the NIST's curves fix $a = -3$. Thus, the point addition requires 12**M**, 4**S**, and 7**A**. Meanwhile, the point doubling uses 4**M**, 4**S**, and 8**A** [Hankerson et al. 2003]. The formulas using Jacobian coordinates are not *complete*; this means that for some particular cases, formulas compute addition law incorrectly. Implementations using incomplete formulas must verify the presence of such special cases and handle them in a proper way. Otherwise, some vulnerabilities can be exploited [Izu and Takagi 2002]. For that reason, the use of complete formulas is recommended.

Renes et al. [Renes et al. 2016] introduced an optimization on the evaluation of complete formulas presented by Bosma and Lenstra [Bosma and Lenstra 1995]. This approach represents points in projective coordinates; i.e. a point $P = (x, y)$ is represented by (X, Y, Z) such that $x = X/Z$ and $y = Y/Z$. For the case of $a = -3$, the point ad-

Algorithm 3 Parallel scheduling of point addition for $a = -3$ using complete formulas.

Input: $P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$, and the coefficient b from $E_{-3,b}(\mathbb{F}_p)$.
Output: $P + Q = (X_3, Y_3, Z_3)$.

	LEFT	RIGHT		LEFT	RIGHT
1:	$l_0 \leftarrow X_1 + Y_1$	$r_0 \leftarrow X_2 + Y_2$	12:	$l_6 \leftarrow l_5 - r_0$	$r_6 \leftarrow r_5$
2:	$l_1 \leftarrow X_1 + Z_1$	$r_1 \leftarrow Y_1 + Z_1$	13:	$l_7 \leftarrow 3l_6$	$r_7 \leftarrow 3r_6$
3:	$l_2 \leftarrow X_2 + Z_2$	$r_2 \leftarrow Y_2 + Z_2$	14:	$l_8 \leftarrow l_0 - r_0$	$r_8 \leftarrow q_0 - r_7$
4:	$p_0 \leftarrow X_1 \times X_2$	$q_0 \leftarrow Y_1 \times Y_2$	15:	$l_9 \leftarrow l_8$	$r_9 \leftarrow q_0 + r_7$
5:	$p_1 \leftarrow l_0 \times r_0$	$q_1 \leftarrow Z_1 \times Z_2$	16:	$l_{10} \leftarrow p_0 + q_0$	$r_{10} \leftarrow q_0 + q_1$
6:	$p_2 \leftarrow l_1 \times l_2$	$q_2 \leftarrow r_1 \times r_2$	17:	$l_{11} \leftarrow p_1 - l_{10}$	$r_{11} \leftarrow q_2 - r_{10}$
7:	$l_0 \leftarrow 3p_0$	$r_0 \leftarrow 3q_1$	18:	$p_4 \leftarrow l_{11} \times r_9$	$q_4 \leftarrow r_{11} \times l_7$
8:	$l_3 \leftarrow p_0 + q_1$	$\emptyset$	19:	$p_5 \leftarrow r_9 \times r_8$	$q_5 \leftarrow l_7 \times l_9$
9:	$l_4 \leftarrow p_2 - l_3$	$\emptyset$	20:	$p_6 \leftarrow l_{11} \times l_9$	$q_6 \leftarrow r_{11} \times r_8$
10:	$p_3 \leftarrow b \times l_4$	$q_3 \leftarrow b \times q_1$	21:	$X_3 \leftarrow p_4 - q_4$	$Y_3 \leftarrow p_5 + q_5$
11:	$l_5 \leftarrow p_3 - p_0$	$r_5 \leftarrow l_4 - q_3$	22:	$Z_3 \leftarrow p_6 + q_6$	$\emptyset$

23: **return** (X_3, Y_3, Z_3)

Algorithm 4 Parallel scheduling of point doubling for $a = -3$ using complete formulas.

Input: $P = (X_1, Y_1, Z_1)$, and the coefficient b from $E_{-3,b}(\mathbb{F}_p)$.
Output: $2P = (X_2, Y_2, Z_2)$.

	LEFT	RIGHT		LEFT	RIGHT
1:	$p_0 \leftarrow X_1 \times X_1$	$q_0 \leftarrow Y_1 \times Y_1$	9:	$l_4 \leftarrow l_0 - l_2$	$r_4 \leftarrow r_1 - r_2$
2:	$p_1 \leftarrow X_1 \times Y_1$	$q_1 \leftarrow Y_1 \times Z_1$	10:	$l_5 \leftarrow 3l_4$	$r_5 \leftarrow r_4$
3:	$p_2 \leftarrow 2X_1 \times Z_1$	$q_2 \leftarrow Z_1 \times Z_1$	11:	$p_4 \leftarrow l_3 \times l_5$	$q_4 \leftarrow r_3 \times r_5$
4:	$p_3 \leftarrow b \times p_2$	$q_3 \leftarrow b \times q_2$	12:	$p_5 \leftarrow p_1 \times r_5$	$q_5 \leftarrow q_1 \times l_5$
5:	$l_0 \leftarrow p_3 - p_0$	$r_0 \leftarrow q_3 - p_2$	13:	$\emptyset$	$q_6 \leftarrow q_1 \times r_1$
6:	$l_1 \leftarrow 3p_0$	$r_1 \leftarrow q_0$	14:	$X_2 \leftarrow 2(p_5 - q_5)$	$Y_2 \leftarrow p_4 + q_4$
7:	$l_2 \leftarrow 3q_2$	$r_2 \leftarrow 3r_0$	15:	$Z_2 \leftarrow 8q_6$	
8:	$l_3 \leftarrow l_1 - l_2$	$r_3 \leftarrow r_1 + r_2$			

16: **return** (X_2, Y_2, Z_2)

dition uses 12**M**, 2$\mathbf{M}_b$, and 29**A**, while the point doubling requires 8**M**, 2$\mathbf{M}_b$, 3**S**, and 21**A**; where $\mathbf{M}_b$ denotes the multiplication by the coefficient b of the elliptic curve. For point additions, Renes' formulas are as efficient as the Jacobian ones; however, there is a considerable amount of field additions.

4.1. Parallel Scheduling of Complete Addition Formulas

By analyzing the Renes' formulas, we notice that is possible to schedule field operations in such a way that two independent operations be executed in parallel. Our proposed schedule is listed in Algorithm 3 for point addition and in Algorithm 4 for point doubling. Assuming the use of two execution units, say LEFT and RIGHT units, the scheduling proceeds as follows. The LEFT unit will compute variables named p_i whenever the operation is a field multiplication; otherwise, they are called l_i. Analogously, the RIGHT unit will compute variables q_i for field multiplications, and r_i for the rest of the operations. Every line of Algorithm 3 (and Algorithm 4) computes pairs (p_i, q_i) or (l_i, r_i) using in most of the cases the same type of field operation. A line with $\emptyset$ symbol denotes no computation in this unit. The communication between units occurs when the LEFT unit uses variables previously computed by the RIGHT unit, and vice versa; we arranged field operations in such a way that minimizes the communication required.

 ©2016 SBC — Soc. Bras. de Computação

We take advantage of the AVX2 vector unit to process field operations simultaneously since the 256-bit vector unit can also behave like two independent vector units of 128 bits. Hence, by making a slight modification in the distribution presented in Figure 2, we store two elements $a, b \in \mathbb{F}_{p384}$ by packaging the digits a_i and b_i in seven 256-bit vector registers as Figure 3 depicts. The techniques for arithmetic operations presented in Section 3.3 are straightforwardly applicable to this representation.

The benefit of using the parallel scheduling proposed is notorious since it reduces by half the amount of operations required to compute point additions. For example, Algorithm 3 takes $6\widehat{\mathbf{M}}$, $1\widehat{\mathbf{M}}_b$, and $15\widehat{\mathbf{A}}$, where $\widehat{\cdot}$ denotes field operations executed in parallel. The communication between units is accomplished by performing permutations of the 128-bit parts that conform a 256-bit vector register using either the `VPERMQ` or the `VPERM2I128` instruction.

5. Elliptic Curve Point Multiplication

We focus on the optimization of three special cases of point multiplication. First, the *variable-point* multiplication, kP, where k represents a secret value, and P is an arbitrary point. Second, the *fixed-point* multiplication, kP, is processed under the assumption that the point P is previously known and fixed, and k is a secret value . Third, the *double-point* multiplication, $kP + lQ$, where P is a fixed point, and Q is an arbitrary point.

5.1. Variable-Point Multiplication

We compute variable-point multiplication using a deterministic execution pattern. First, we implemented the algorithm employed by Bos et al. [Bos et al. 2015, Alg. 1] and reproduced in Algorithm 5. For this case, we set $\omega = 6$, which implies to calculate a series of five point doublings followed by one point addition. Since the fixed-window recoding works only for odd numbers, we recode either k (when odd) or $\ell - k$ (when even), and at the end of the algorithm, we must choose between kP or $-kP$. We remark that both selections require running in constant time; hence, we use Algorithm 7 for meeting this requirement.

In line 8 of Algorithm 5, we perform queries, denoted by ϕ, to the look-up table according to the digits k'_i; to do that securely, we read the entire table and conditionally select the appropriate entry. The secure query (Algorithm 6) uses two primitives to operate: conditional move operation (Algorithm 8) and selection operation (Algorithm 7). We implement these functions using logic arithmetic over vector registers; this avoids that running time depends on the values of the operands reducing the success chances of a timing attack.

5.2. Fixed-Point Multiplication

In this case, the point multiplication can be accelerated by generating a look-up table containing some precomputed points. Since it is allowed to produce a large look-up table, this scenario exhibits a trade-off between speed and memory footprint. In our implementation, the fixed-point multiplication algorithm read the entire look-up table; hence, it would be desirable that a large part of the table fit on L1-D (data memory cache), which has a capacity of 32 KiB in the Intel Core i7 processors.

 ©2016 SBC — Soc. Bras. de Computação

Algorithm 5 Variable-point multiplication using a deterministic pattern.

Input: P, k, and ω, where $P \in E$, k is a positive integer lesser than ℓ, and ω is a positive integer.
Output: Q, a point such that $Q = kP$.

1: $t \leftarrow \lceil \log_2(\ell)/(\omega - 1) \rceil$
2: $T = \{(2i+1)P \mid \text{for } i \in [0, 2^{\omega-2})\}$
3: $k' \leftarrow \texttt{Select}(\ell - k, k, k \bmod 2, 0)$
4: $(k'_0, \ldots, k'_{t-1}) \leftarrow \texttt{Recoding}(k', \omega)$
5: $Q \leftarrow \mathcal{O}$
6: **for** $i \leftarrow t-1$ **to** 0 **do**
7: $\quad Q \leftarrow 2^{\omega-1}Q$
8: $\quad Q \leftarrow Q + \phi(T, k'_i)$ *(Alg. 6.)*
9: **end for**
10: $Q_y \leftarrow \texttt{Select}(-Q_y, Q_y, k \bmod 2, 0)$
11: **return** Q

Algorithm 6 Secure look-up table query ϕ.

Input: T and v, where T is a look-up table storing d points; and v is an integer stored into a n-bit word.
Output: R, a point in projective coordinates such that $R = T_v$.

1: $(x, y) \leftarrow (0, 0)$
2: $(V, S) \leftarrow \left(\lfloor \frac{v+1}{2} \rfloor, v \gg (n-1)\right)$
3: **for** $i \leftarrow 1$ **to** d **do**
4: $\quad (x_T, y_T) \leftarrow T_i$
5: $\quad x \leftarrow x \oplus \texttt{CMove}(x_T, i, V)$ *(Alg. 8.)*
6: $\quad y \leftarrow y \oplus \texttt{CMove}(y_T, i, V)$ *(Alg. 8.)*
7: **end for**
8: $y \leftarrow \texttt{Select}(1, 0, V, 0)$ *(Alg. 7.)*
9: $z \leftarrow \texttt{Select}(0, 1, V, 0)$ *(Alg. 7.)*
10: $y \leftarrow \texttt{Select}(y, -y, S, 0)$ *(Alg. 7.)*
11: **return** $R \leftarrow (x, y, z)$

Algorithm 7 `Select`: Secure selection operation.

Input: A, B, x, and y, where A and B are two k-bit integer numbers; and, x and y are two n-bit integers.
Output: If $x = y$, returns A; otherwise, returns B.

1: $b \leftarrow (-(x \wedge y)) \gg (n-1)$
2: $M \leftarrow (\underbrace{bbb\cdots}_{n \text{ times}})_2$
3: **return** $(\neg M \wedge A) \oplus (M \wedge B)$

Algorithm 8 `CMove`: Secure conditional move operation.

Input: A, x, and y, where A is a k-bit integer number; and, x and y are two n-bit integers.
Output: If $x = y$, returns A; otherwise, returns 0.

1: $b \leftarrow (-(x \wedge y)) \gg (n-1)$
2: $M \leftarrow (\underbrace{bbb\cdots}_{n \text{ times}})_2$
3: **return** $\neg M \wedge A$

Initially, we precompute a look-up table with entries $T_i = \{j2^{4i}P\}$ for all even $i \in [0, 96)$ and all $0 \leq j \leq 8$. For computing kP, first, we split scalar k in 96 signed digits of four bits, such that $k = \sum_{i=0}^{95} k_i 2^{4i}$. Then, kP is obtained as $kP = Q_0 + 2^4 Q_1$, where $Q_0 = \sum_{i=0}^{95} \phi(T_i, k_i)$ for $i \equiv 0 \bmod 2$, and $Q_1 = \sum_{i=0}^{95} \phi(T_{i-1}, k_i)$ for $i \equiv 1 \bmod 2$. This computation requires in total 96 point additions and 4 point doublings; whereas regarding memory footprint, the look-up table stores 384 points in affine coordinates accounting for 36 KiB of read-only memory; what is close to the size of L1-D. Like in the case of variable-point multiplication, the queries are securely processed by Algorithm 6.

5.3. Double-Point Multiplication

The $kP + lQ$ operation can be computed using the interleaved algorithm together with ω-NAF representation [Hankerson et al. 2003, Alg. 3.51]. This algorithm encodes both scalars k and l into ω-NAF expansions [Solinas 2000]. Indeed, the value of the window-ω can be different for each scalar; thus, let ω_k and ω_l be the window values selected for the scalar k and l, respectively. The value ω_k impacts significantly on both the running time

 ©2016 SBC — Soc. Bras. de Computação

and the memory footprint of the implementation, whereas the ω_l value impacts only on the running time. In our implementation, we choose $\omega_k = 7$ and $\omega_l = 5$ since these values minimize the running time of the double-point multiplication.

6. Benchmarking Results

To evaluate the impact on the performance of our optimizations, we measure the time to compute the shared secret in the ECDH protocol, the signature generation, and the verification operation of the ECDSA scheme. Then, we make a comparison against other publicly-available cryptographic libraries. We choose libraries mainly written in C-language that offer support to the P-384 curve, and we end up with the following:

- OpenSSL (v1.0.2h) is the state-of-the-art on open-source cryptographic implementations; this library is in continuous development containing a vast number of optimizations [The OpenSSL Project 2003].
- Nettle (v3.2) is a low-level cryptographic library that provides a back-end functionality to the GnuTLS library [Niels Möller 2001].
- mbed TLS (v2.2.1), previously known as PolarSSL, is a multi-platform library with the aim to ease the development of cryptographic algorithms [Bakker 2008].
- Relic toolkit (v0.4.1) is a modern library that supports a broad range of cryptographic algorithms. Measurements were done using the default configurations, and setting GMP as the arithmetic backend [Aranha and Gouvêa 2009].
- BoringSSL (commit `fe47ba2f`) is a fork of the OpenSSL library modified with the aim to cover the requirements needed by Google's products [Google 2015].

Table 2 shows the performance comparison obtained. As can be seen, the cryptographic libraries selected offer similar performance for the P-348 curve, except the mbed TLS library, which is by far the slowest. In all scenarios, our software implementation outperforms the timings of the other libraries. We want to note that these libraries still compute the point addition using the Jacobian formulas, which present special cases that must be handled in constant time, and this could add a considerable performance penalty.

The authors of complete point addition formulas presented a proof-of-concept implementation of their formulas using the OpenSSL library [Renes et al. 2016]; they observed a slow-down factor of $1.41\times$ for computing the ECDH protocol. We state that this factor can be reduced by using the implementation techniques presented in this work. For example, we benchmark the prime field arithmetic operations and notice that our implementation computes modular additions $3\times$ faster than the OpenSSL library, which is a relevant result since the complete formulas require lots of prime field additions; thus, our implementation reduces the overhead caused by the complete formulas.

Regarding differences in the performance observed on the Haswell and the Skylake micro-architectures, our implementation had better performance improvement that the other libraries. For example, in the shared secret computation, the running time of our software is around 10% faster when is executed on Skylake, whereas the other libraries are accelerated only in 5%. A plausible explanation of this situation is that the improvements in the latest micro-architectures have a higher impact on vector instructions rather than the native scalar instructions. Therefore, it is expected that vectorized code will have a greater performance in the forthcoming micro-architectures.

 ©2016 SBC — Soc. Bras. de Computação

Table 2. Performance of ECC protocols using the P-384 curve. Entries represent 10^6 clock cycles measured on a Haswell (Core i7-4770) and on a Skylake (Core i7-6700K) micro-architectures. All libraries were compiled using the GNU Compiler Collection (v5.3.1) and executed with the Intel Turbo Boost and the Intel Hyper-Threading technologies disabled.

Cryptographic library	Haswell			Skylake		
	ECDH	ECDSA		ECDH	ECDSA	
	Shared secret	Signature	Verification	Shared secret	Signature	Verification
mbed TLS	18.70	7.15	26.49	17.90	6.72	25.25
BoringSSL	3.60	3.78	4.47	3.43	3.67	4.33
Relic toolkit	1.79	0.89	2.40	1.52	0.77	2.06
Nettle	—	0.77	2.07	—	0.62	1.57
OpenSSL	2.12	0.65	2.60	2.03	0.62	2.49
This work	1.25	0.56	1.31	1.11	0.53	1.11

7. Concluding Remarks

Given the recent discovery of the efficient evaluation of the complete formulas for point addition, we contributed with a new parallel scheduling of field operations that allows computing two independent operations simultaneously. This technique works either for hardware or software implementations. Moreover, the scheduling is not restricted to the P-384 curve, since by applying slight modifications, it applies to all prime elliptic curves in short Weierstrass form.

Additionally, we presented software techniques for implementing the arithmetic operations of $\mathbb{F}_{p_{384}}$ using vector instructions. Essentially, we packed pairs of field elements into 256-bit vector registers, and we arranged vector instructions to perform two field operations at the same time. These operations served as building blocks for the implementation of the parallel scheduling of the complete formulas for point addition. In fact, we optimized our implementation to make an efficient use of the execution engine for micro-architectures supporting the AVX2 instruction set.

The widespread use of ECC can be hampered by low-performance implementations provided by some cryptographic libraries. In this work, the benchmark results revealed that the use of parallel execution units improves the performance of ECC. Specifically, we observed that the combination of the parallel scheduling proposed together with the efficient computation of field arithmetic allowed to obtain a significant reduction in the running time of the ECDSA scheme and the ECDH protocol using the P-384 curve. Finally, we expect that the techniques presented in this work contribute to improving the performance of elliptic curves on processors that support vector instructions.

Acknowledgments. We thank the anonymous reviewers for their helpful comments.

References

[ANSI 1999] ANSI (1999). ANS X9.62 Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).

[ANSI 2001] ANSI (2001). ANS X9.63 Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.

 ©2016 SBC — Soc. Bras. de Computação

[Aranha and Gouvêa 2009] Aranha, D. F. and Gouvêa, C. P. L. (2009). RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit.

[Bakker 2008] Bakker, P. (2008). mbed TLS. (v2.3). https://tls.mbed.org/.

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

[Bos et al. 2015] Bos, J. W., Costello, C., Longa, P., and Naehrig, M. (2015). Selecting elliptic curves for cryptography: an efficiency and security analysis. *Journal of Cryptographic Engineering*, pages 1–28.

[Bos et al. 2014] Bos, J. W., Halderman, J. A., Heninger, N., Moore, J., Naehrig, M., and Wustrow, E. (2014). *Elliptic Curve Cryptography in Practice*, pages 157–175. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bosma and Lenstra 1995] Bosma, W. and Lenstra, H. (1995). Complete Systems of Two Addition Laws for Elliptic Curves. *Journal of Number Theory*, 53(2):229–240.

[Brown 2009] Brown, D. R. L. (2009). SEC 1: Elliptic Curve Cryptography. http://www.secg.org/sec1-v2.pdf.

[CNSS 2012] CNSS (2012). National Information Assurance Policy on the Use of Public Standards for the Secure Sharing of Information Among National Security Systems. CNSSP Policy 15. https://www.cnss.gov/CNSS/issuances/Policies.cfm.

[CNSS 2015] CNSS (2015). Use of Public Standards for the Secure Sharing of Information among National Security Systems. CNSS Advisory Memorandum 02-15. https://www.cnss.gov/CNSS/issuances/Memoranda.cfm.

[Google 2015] Google (2015). BoringSSL. https://boringssl.googlesource.com/boringssl/+/fe47ba2fc5512436696f745b5756d08c7d8ceb0b.

[Granger and Scott 2015] Granger, R. and Scott, M. (2015). Faster ECC over $\mathbb{F}_{2^{521}-1}$. In Katz, J., editor, *Public-Key Cryptography – PKC 2015: 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 – April 1, 2015, Proceedings*, pages 539–553, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Gueron and Krasnov 2014] Gueron, S. and Krasnov, V. (2014). Fast prime field elliptic-curve cryptography with 256-bit primes. *Journal of Cryptographic Engineering*, pages 1–11.

[Hankerson et al. 2003] Hankerson, D., Menezes, A. J., and Vanstone, S. (2003). *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[IEEE 2000] IEEE (2000). IEEE Standard Specifications for Public-Key Cryptography. Number 1363, pages 1–228. IEEE Std 1363-2000.

[Intel Corporation 2011] Intel Corporation (2011). Intel® Advanced Vector Extensions Programming Reference. Technical report. https://software.intel.com/sites/default/files/m/f/7/c/36945.

 ©2016 SBC — Soc. Bras. de Computação

[Izu and Takagi 2002] Izu, T. and Takagi, T. (2002). Exceptional Procedure Attack on Elliptic Curve Cryptosystems. In Desmedt, Y. G., editor, *Public Key Cryptography — PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings*, pages 224–239, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Käsper 2012] Käsper, E. (2012). Fast Elliptic Curve Cryptography in OpenSSL. In Danezis, G., Dietrich, S., and Sako, K., editors, *Financial Cryptography and Data Security*, volume 7126 of *Lecture Notes in Computer Science*, pages 27–39. Springer Berlin Heidelberg.

[Koblitz 1987] Koblitz, N. (1987). Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209.

[Miller 1986] Miller, V. S. (1986). Use of Elliptic Curves in Cryptography. In Williams, H. C., editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer Berlin Heidelberg.

[Niels Möller 2001] Niels Möller (2001). Nettle. `http://www.lysator.liu.se/~nisse/nettle`.

[NIST 1999] NIST (1999). Recommended elliptic curves for federal government use.

[NIST 2000] NIST (2000). Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-2. `http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf`.

[Pollard 1975] Pollard, J. (1975). A monte carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334.

[Proos and Zalka 2003] Proos, J. and Zalka, C. (2003). Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves. *Quantum Information & Computation*, 3(4):317–344.

[Renes et al. 2016] Renes, J., Costello, C., and Batina, L. (2016). Complete Addition Formulas for Prime Order Elliptic Curves. In Fischlin, M. and Coron, J.-S., editors, *Advances in Cryptology – EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 403–428, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Rivest et al. 1978] Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

[Solinas 2000] Solinas, J. (2000). Efficient Arithmetic on Koblitz Curves. *Designs, Codes and Cryptography*, 19(2-3):195–249.

[Solinas 1999] Solinas, J. A. (1999). Generalized Mersenne Numbers. Technical Report CORR 99-39, Center of Applied Cryptographic Research (CACR).

[The OpenSSL Project 2003] The OpenSSL Project (2003). OpenSSL: The Open Source toolkit for SSL/TLS. `http://www.openssl.org`.

[Vanstone et al. 1992] Vanstone, S., Rivest, R. L., Hellman, M. E., Anderson, J. C., and Lyons, J. W. (1992). Responses to NIST's Proposal. *Communications of the ACM*, 35(7):41–54. (John Anderson communicated Vanstone's proposal).

 ©2016 SBC — Soc. Bras. de Computação