Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 15 (10): 1261-1265, 2015
ISSN 1812-5654

RESEARCH ARTICLE OPEN ACCESS

DOI: 10.3923/jas.2015.1261.1265

Highly Efficient Generic-Point Parallel Scalar Multiplication using Concurrent Precomputations

Turki F. Al-Somani
Department of Computer Engineering, Umm Al-Qura University, P.O. Box 715, Makkah, 21955, Saudi Arabia

ARTICLE INFO
Article History:
Received: August 07, 2015
Accepted: October 01, 2015

Corresponding Author:
Turki F. Al-Somani
Department of Computer Engineering, Umm Al-Qura University, P.O. Box 715, Makkah, 21955, Saudi Arabia

ABSTRACT
An efficient precomputation-based generic-point parallel scalar multiplication method is presented in this study. The proposed method takes a set of generic points and performs their precomputations concurrently using an equivalent set of elliptic curve cryptoprocessors. Then, parallel scalar multiplications are performed sequentially for each of the generic points. The new method outperforms the existing postcomputation-based methods. Furthermore, it is scalable for any number of parallel processors and performs better as the number of consecutive requests increases. The proposed method has been implemented on an FPGA and its Area-Time^2 (AT^2) performance metric outperformed recent fast implementations. Accordingly, the proposed method is very suitable for use in high-performance end servers that use parallel elliptic curve processors.

Key words: Elliptic curves cryptosystems, parallel scalar multiplication, precomputations, postcomputations

INTRODUCTION

Initially proposed by Miller (1986) and Koblitz (1987), Elliptic Curve Cryptosystems (ECCs), which are based on the Discrete Logarithm Problem (DLP), are perceived to be a serious alternative to the RSA system with a much shorter word length (Rivest *et al*., 1978). Due to the apparent difficultly in tackling the problem, the key sizes can be considerably reduced (Blake *et al*., 1999) and the security of an ECC with a key size of 128-256 bits has been proven to be equal to that of an RSA system with 1-2 kb. Due to these advantages, ECCs have recently gained considerable recognition andare incorporated in many standards.

The basic operation for ECCs is scalar multiplication, denoted as kP (Hankerson *et al*., 2004). Scalar multiplication of a group of points on an elliptic curve is analogous to the exponentiation of a multiplicative group of integers modulo a fixed integer m. However, for high-performance end servers, the current sequential scalar multiplication methods are too slow to meet the demands of the increasing numbers of customers. Thus, efficient scalar multiplication methods for such servers need to be identified. Scalar multiplication methods that can be parallelized are often used for high-speed implementations. Although precomputations (Brickell *et al*., 1993) have been applied to speed up scalar multiplication, they require sequential steps that cannot be parallelized and are primarily advantageous when the elliptic curve point is fixed. However, during secure communication sessions using public keys, the elliptic curve point changes, as it depends on the public key of the communicating entity and hence, session dependent. This is also the case when digital signatures are used. Therefore, scalar multiplications are generally performed with a generic elliptic curve point. Because the elliptic curve point is likely to differ in each session, the overheads resulting from the necessary precomputations must be considered when estimating the total computational time required.

Postcomputations have recently been proposed (Al-Somani and Ibrahim, 2009; Al-Somani, 2010; Al-Somani *et al*., 2014) as an alternative method of speeding up scalar multiplication. In (Al-Somani and Ibrahim, 2009), the precomputation overheads are replaced by postcomputations that can be parallelized. The multiplier k is partitioned into u partitions that can be processed in parallel by u processors using the binary method. Postcomputations are then distributed on u-1 processors to be performed in parallel. The points that result from processing these key partitions with the postcomputations are finally assimilated to produce kp. However, the performance of the postcomputation-based method is analyzed in (Al-Somani, 2010) and the results show that the best performance is achieved when 8 cryptoprocessors are used with $128 \le m \le 256$, which limits the performance when more parallel cryptoprocessors are available.

An efficient postcomputation method was recently proposed in (Al-Somani *et al*., 2014), which uses an efficient mapping technique between the computations and the processors. The results in (Al-Somani *et al*., 2014) show that the proposed method is efficient and scalable for any number of parallel processors and performs better as the number of consecutive requests increases. In this study, a highly efficient precomputation-based generic-point parallel scalar multiplication method is presented.

BRIEF INTRODUCTION TO ELLIPTIC CURVE CRYPTOSYSTEMS

Elliptic curve cryptosystems (Miller, 1986; Koblitz, 1987) attract a great deal of research attention and are included in numerous standards. The ECCs are emerging as an attractive alternative to other public-key schemes such as RSA as they offer a smaller key size and higher strength per bit. Extensive research has been conducted on the underlying mathematics, security strength and efficient implementation of ECCs. The discrete points on an elliptic curve form an abelian group, the group operation of which is known as "Point addition" (PADD). Elliptic curve point addition is defined according to the "Chord-tangent process." Point addition of the same point to itself is referred to as point doubling (PDBL).

Scalar multiplication is the basic operation in ECC (Hankerson *et al*., 2004). Computing kP can be achieved with the binary method (Hankerson *et al*., 2004), the so called double and add method, based on the binary expression of the multiplier k. The kP can be computed using a binary method as follows.

Let $k = (k_{m-1},\ldots,k_0)$, where k_{m-1} is the most significant bit of k, be the binary representation of k. The multiplier k can be written as:

$$k = \sum_{0 \le i < m} k_i 2^i = k_{m-1} 2^{m-1} + \cdots + k_1 2 + k_0 \quad (1)$$

Using the Horner expansion, k can be rewritten as:

$$k = (\cdots((k_{m-1}2+k_{m-2})2+\cdots+k_1)2+k_0) \quad (2)$$

Accordingly:

$$kP = 2\,(\cdots 2\,(2k_{m-1}P+k_{m-2}P)+\cdots+k_1P+k_0P \quad (3)$$

The average time complexity of the binary method is:

$$\text{Time complexity} = (m)\,\text{PDBL}+(m/2)\,\text{PADD} \quad (4)$$

PROPOSED METHODS

The main idea here is to perform u sequential precomputations of u generic points concurrently using u processors in the first stage by mapping each generic point to an individual processor. In the second stage, the precomputed points are used to perform the parallel scalar multiplication of each point of the u generic points. Each multiplier k of the u multipliers is partitioned into a number of equally sized partitions (v) that can be processed in parallel by u processors. The points resulting from processing these key partitions are assimilated at the end to produce kP. The number of available processors limits the number of partitions of each k. For a particular P and k, each partition is associated with a precomputed point to maintain its significance. For u partitions, (u-1) precomputed points are required for each k. Precomputed points can be computed simply by a sequence of doubling operations of the base point P. The computation of kP for a particular P and k in parallel can be carried out as follows.

Let $k = (k_{m-1},\ldots, k_0)$ be the binary representation of multiplier k. Let u be a small integer and let v = (m/u). A set of (uv-m) zero bits is appended to the left of the binary representation of k and the resulting bit string is partitioned into u partitions each of a length v. Thus, k becomes $k = (k^{(u-1)}||k^{(u-2)}||\ldots||k^{(1)}||k^{(0)})$, where, each $k^{(i)}$ is a partition of a length v = (m/u). The scalar multiplication product kp can be computed as:

$$kP = \sum_{0 \le i \le u} s_i \quad (5)$$

where, s_i is defined as:

$$s_i = 2\,(\cdots 2\,(2k_{iv+v-1}(2^{iv}P)+k_{iv+v-2}(2^{iv}P))+\cdots+k_{iv+1}(2^{iv}P))+k_{iv+0}(2_{iv}P) \quad (6)$$

The kP for u points and keys using Eq. 5 and 6 can be computed using the following algorithm.

Algorithm 1: Proposed method

Step 1:	**Inputs:** P[0], P[1],..., P[u-1], k[0], k[1],..., k[u-1]
Step 2:	By padding the ks with (uv-m) zeros if necessary, write each k = ($k^{(u-1)}$‖$k^{(u-2)}$‖…‖$k^{(1)}$‖$k^{(0)}$), where each $k^{(i)}$ is a partition of length v = [m/u]
Step 3:	Initialization
Step 3.1:	For i = 0 to u-1, do
Step 3.1.1:	Q[i]←P[i]
Step 3.1.2:	R[i]←O
Step 4:	Perform concurrent precomputations of u points for each of the Ps
Step 4.1:	For i = 0 to u-1, do (in parallel)
Step 4.1.1:	P_0[i]←Q[i]
Step 4.2:	For w = 0 to u-1 do (in parallel)
Step 4.2.1:	for i = 1 to u-1, do
Step 4.2.1.1:	for j = 0 to v-1, do
Step 4.2.1.1.1:	Q[w]←2Q[w]
Step 4.2.1.2:	P_i[w]←Q[w]
Step 5:	Key partition association with precomputed points
Step 5.1:	for i = 0 to u-1 do
Step 5.1.1:	for j = 0 to u-1 do
Step 5.1.1.1:	($k[i]^{(j)}$, P_j[i]).
Step 6:	Scalar multiplication:
Step 6.1:	for i = 0 to u-1 do
Step 6.1.1:	for j = 0 to u-1, do (in parallel)
Step 6.1.1.1:	Q[i] The Binary Method ($k[i]^{(j)}$, P_j[i])
Step 6.1.1.2:	R[i]←R[i]+Q[i]
Step 6.1.2:	**Output** R[i]

Algorithm 1 shows the pseudo code of the proposed method. Algorithm 1 accepts u requests, which means u points and u keys. For u requests, P[i] and k[i] represent the base point and the key of the request i, respectively. The partitioning of the ks into u partitions is performed in Step 2. For a particular key k[i], $k[i]^{(j)}$ represents the jth partition of the key k[i]. Precomputed points are computed concurrently for each of the P points by repeated doubling in Step 4. For a particular point P[i], P_j[0] represents the base point of the key partition J.

For a particular P and k, each partition $k^{(i)}$ is associated with a particular precomputed point P_i to maintain the significance of each partition (Step 5). For a particular P and k, parallel scalar multiplications start in Step 6. An individual processor processes each partition independently. The points resulting from each execution of the binary scalar multiplication method (Hankerson *et al*., 2004) are accumulated in point R (Step 6.1.1.2), which requires (u-1) the addition of extra points.

Example: Let, k = $(1000\ 0101\ 1100\ 0011)_2 = (34243)_{10}$, m = 16 u = 2. The key partitions are $k^{(0)}$ = 1100 0011 and $k^{(1)}$ = 1000 0101. The scalar multiplication of these partitions is then computed in parallel for two consecutive requests, using the same key for simplicity and two different points (P[0] and P[1]), as:

Stage 1: Concurrent precomputations stage.

Processor$_{(0)}$: Precompute P_8[1]

Processor$_{(1)}$: Precompute P_8[1]

Stage 2: Processing stage

- Processing the 1st request (kP_0)

Processor$_{(0)}$: $s_{0,P[0]}$ = (2(2(2(2(2(2(2(1)-P[0]+(1)-P[0])+(0)P[0])+(0)P[0])+(0)P[0])+(0)P[0])+(1)P[0])+(1)P[0]) = 195 P[0]

Processor$_{(1)}$: $s_{1,P[0]}$ = (2(2(2(2(2(2(2(1)-P_8[0]+(0)P_8[0])+(0)P_8[0])+(0)P_8[0])+(0)P_8[0])+(1)P_8[0])+(0)P_8[0])+(1)P_8[0]) = 34048 P_8-[0]

Accordingly, kP[0] is computed as:

$$kP[0] = s_{0,P[0]}+s_{1,P[0]} = 195P[0]+34048P[0] = 34243P[0]$$

- Processing the 2nd request (kP[1])

Processor$_{(0)}$: $s_{0,P[1]}$ = (2(2(2(2(2(2(2(1)-P[1]+(1)-P[1])+(0)P[1])+(0)P[1])+(0)P[1])+(0)P[1])+(1)P[1])+(1)P[1]) = 195 P[1]

Processor$_{(1)}$: $s_{1,P[1]}$ = (2(2(2(2(2(2(2(1)-P_8[1]+(0)P_8[1])+(0)P_8[1])+(0)P_8[1])+(0)P_8[1])+(1)P_8[1])+(0)P_8[1])+(1)P_8[1]) = 34048 P-[1]

Accordingly, kP[1] is computed as:

$$kP[1] = s_{0,P[1]}+s_{1,P[1]} = 195P[1]+34048P[1] = 34243P[1]$$

PERFORMANCE ANALYSIS

The average time the proposed method takes to compute kP, depending on the number of consecutive requests, denoted here by rs, equals:

$$\text{Time complexity} = \left(\frac{r}{u}\right)\left(\left(v(u-1)\,DBL\right)\right) + r\,(vDBL + \left(\frac{v}{2}+u-1\right)ADD \quad (7)$$

The number of precomputed points to be stored for each of the u generic points equals (u-1). Each partition requires one storage element, as in the binary method. Accordingly, the number of storage elements required for the proposed method equals u(u-1)+u = u^2 storage elements. However, space is not a limiting factor here, as the target is high-performance end servers that use parallel processors.

The performance analysis results in (Al-Somani, 2010) show that the best performance is achieved when eight processors are used. Accordingly, the methods in (Al-Somani and Ibrahim, 2009; Al-Somani *et al*., 2014) are compared with the method proposed in this study when eight processors (u = 8) are deployed using different values of m. For simplicity, it is assumed here that the required computation time for point addition is twice that required for point doubling. Table 1 shows the comparison results, in terms of point doublings, with up to 32 consecutive requests. Figure 1 depicts the results when m = 256 bits.

Table 1: Results of the proposed method with m = 128, 160, 200, 256 and u = 8

	Total (DBLs)														
	r = 2			r = 4			r = 8			r = 16			r = 32		
m	*	**	New	*	**	New	*	**	New	*	**	New	*	**	New
128	268	190	120.00	536	380	240.0	1072	760	480	2144	1520	960	4288	3040	1920
160	332	230	143.00	664	460	286.0	1328	920	572	2656	1840	1144	5312	3680	2288
200	412	280	171.75	824	560	343.5	1648	1120	687	3296	2240	1374	6592	4480	2748
256	530	350	212.00	1060	700	424.0	2120	1400	848	4240	2800	1696	8480	5600	3392

*Al-Somani and Ibrahim (2009) and **Al-Somani *et al*. (2014)

Table 2: AT^2 comparison results over GF (2^{163})

		AT^2						
Area (slices)	Time (μsec)	r = 1	r = 2	r = 4	r = 8	r = 16	r = 32	References
15020	36.77	2.03E+07	8.12E+07	3.25E+08	1.30E+09	5.20E+09	2.08E+10	Chelton and Benaissa (2006)
15368	33.05	1.68E+07	6.71E+07	2.69E+08	1.07E+09	4.30E+09	1.72E+10	Chelton and Benaissa (2008)
4080	20.56	1.72E+06	6.90E+06	2.76E+07	1.10E+08	4.42E+08	1.77E+09	Ansari and Hasan (2008)
20807	7.72	1.24E+06	4.96E+06	1.98E+07	7.94E+07	3.17E+08	1.27E+09	Zhang *et al*. (2010)
22936	5.48	6.89E+05	2.76E+06	1.10E+07	4.41E+07	1.76E+08	7.05E+08	Sutter *et al*. (2013)
62073	2.75	4.68E+05	1.87E+06	7.49E+06	3.00E+07	1.20E+08	4.80E+08	New (proposed here)

A: Area, T: Time

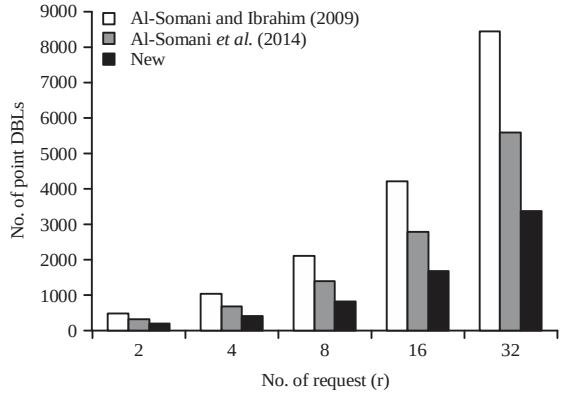

Fig. 1: Implementation results with m = 256 and u = 8

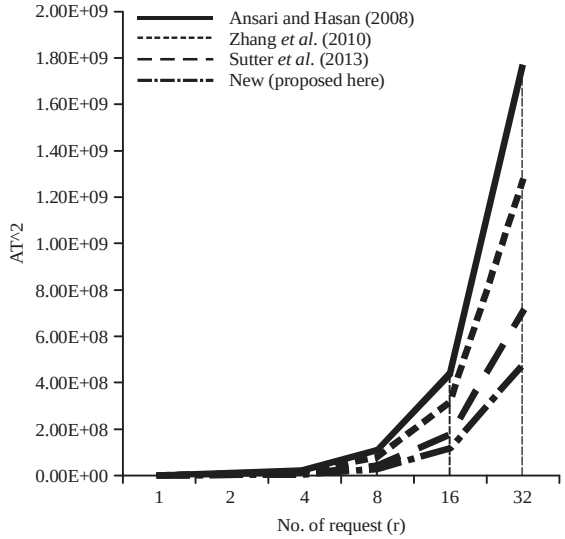

Fig. 2: Results with m = 256 and u = 8

RESULTS

The lower bound on the area-time cost of a given design is usually used as a performance metric $AT^{2\alpha}$, $0 \leq \alpha \leq 1$, where the choice of α determines the relative importance of area and time (Thompson, 1980). Such lower bounds have been obtained for several problems, e.g., discrete Fourier transform, matrix multiplication, binary addition and others (Thompson, 1980). Once the lower bound on the chosen performance metric is known, designers attempt to devise algorithms and designs that are optimal for a range of area and time values. Even though a design might be optimal for area (A) and time (T) values within a certain range, it is nevertheless of interest to obtain designs for the minimum time, i.e., maximum speed performance and area values.

In this study, we are interested in high performance implementations. Accordingly, to make a more meaningful comparison between the proposed method and recent fast implementations (Chelton and Benaissa, 2006; Ansari and Hasan, 2008; Chelton and Benaissa, 2008; Zhang *et al*., 2010; Sutter *et al*., 2013), the AT^2 measure is evaluated. The proposed method was modeled using VHDL and synthesized on a Xilinx Vertex 7 FPGA (xc7v2000t-2flg1925) over GF (2^{163}) with Gaussian Normal Basis (Ash *et al*., 1989) and Lopez-Dahab coordinates (Lopez and Dahab, 1998). The proposed ECC cryptoprocessors consists of eight scalar multiplication processors. The synthesis report shows that the number of slices is 62073 and the maximum frequency is 297.461 MHZ. Table 2 shows the AT^2 comparison results for the proposed method and recent fast implementations with up to 32 consecutive requests. Figure 2 depicts the results of the four fastest implementations.

DISCUSSION

Table 1 shows the comparison results for the proposed method and the postcomputation methods (Al-Somani and Ibrahim, 2009; Al-Somani *et al*., 2014). The results clearly show that the proposed method outperforms the postcomputation methods (Al-Somani and Ibrahim, 2009; Al-Somani *et al*., 2014) when the number of consecutive requests is two or more. The results also show that the proposed method performs better as the number of consecutive requests increases. Moreover, the proposed method is scalable for any number of parallel processors and is not limited to just eight, as in (Al-Somani and Ibrahim, 2009).

Table 2 shows the AT^2 comparison results for the proposed method and recent fast implementations (Chelton and Benaissa, 2006; Ansari and Hasan, 2008; Chelton and Benaissa, 2008; Zhang *et al*., 2010; Sutter *et al*., 2013). The results clearly show that the AT^2 performance metric of the proposed implementation outperforms the previous implementations in the literature (Chelton and Benaissa, 2006; Ansari and Hasan, 2008; Chelton and Benaissa, 2008; Zhang *et al*., 2010; Sutter *et al.*, 2013). Accordingly, the proposed method is highly suited for use in high-performance end servers that use parallel elliptic curve processors.

CONCLUSION

Postcomputations have recently been proposed in which the precomputation overheads are replaced by postcomputations that can be parallelized. An efficient precomputation-based generic-point parallel scalar multiplication method is presented in this study. The proposed method accepts a set of generic points and performs their precomputations concurrently using an equivalent set of elliptic curve cryptoprocessors. Then, parallel scalar multiplications are performed sequentially for each of these generic points. The new method outperforms the existing postcomputation-based methods and is scalable for any number of parallel processors. Furthermore, the proposed method was implemented on an FPGA and the AT^2 performance metric of the proposed method outperformed the recent fast implementations in the literature.

ACKNOWLEDGMENT

The author acknowledges the support of both Umm Al-Qura University and King Abdulaziz City for Science and Technology.

REFERENCES

Al-Somani, T.F. and M.K. Ibrahim, 2009. Generic-point parallel scalar multiplication without precomputations. IEICE Electron. Exp., 6: 1732-1736.

Al-Somani, T.F., 2010. Performance analysis of the postcomputation-based generic-point parallel scalar multiplication method. Global J. Comput. Sci. Technol., 10: 32-37.

Al-Somani, T.F., A.G. Fayoumi and M.K. Ibrahim, 2014. An efficient and scalable postcomputation-based generic-point parallel scalar multiplication method. EICE Electron. Express, 11: 1-6.

Ansari, B. and M.A. Hasan, 2008. High-performance architecture of elliptic curve scalar multiplication. IEEE Trans. Comput., 57: 1443-1453.

Ash, D.W., I.F. Blake and S.A. Vanstone, 1989. Low complexity normal bases. Discrete Applied Mathmat., 25: 191-210.

Blake, I.F., G. Seroussi and N.P. Smart, 1999. Elliptic Curves in Cryptography. Cambridge University Press, Cambridge, UK., ISBN-13: 9780521653749, Pages: 204.

Brickell, E.F., D.M. Gordon, K.S. McCurley and D.B. Wilson, 1993. Fast Exponentiation with Precomputation. In: Advances in Cryptology-Eurocrypt'92, Rueppel, R.A. (Ed.). Springer, New York, USA., ISBN-13: 9783540475552, pp: 200-207.

Chelton, W. and M. Benaissa, 2006. High-speed pipelined EGG processor on FPGA. Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation, October 2-4, 2006, Banff, Alta, pp: 136-141.

Chelton, W.N. and M. Benaissa, 2008. Fast elliptic curve cryptography on FPGA. IEEE Trans. Very Large Scale Integrat Syst., 16: 198-205.

Hankerson, D., A. Menezes and S. Vanstone, 2004. Guide to Elliptic Curve Cryptography. 1st Edn., Springer-Verlag, New York, ISBN: 0-387-95273-X.

Koblitz, N., 1987. Elliptic curve cryptosystems. Math. Comput., 48: 203-209.

Lopez, J. and R. Dahab, 1998. Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n). Springer-Verlag, Berlin, pp: 201-212.

Miller, V.S., 1986. Use of Elliptic Curves in Cryptography. In: Advances in Cryptology-CRYPTO'85, Williams, H.C. (Ed.)., Springer-Verlag, Berlin, Heidelberg, pp: 417-426.

Rivest, R.L., A. Shamir and L. Adleman, 1978. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM., 21: 120-126.

Sutter, G.D., J. Deschamps and J.L. Imana, 2013. Efficient elliptic curve point multiplication using digit-serial binary field operations. IEEE Trans. Ind. Electron., 60: 217-225.

Thompson, C.D., 1980. A complexity theory for VLSI. Ph.D. Thesis, Department of Computer Science, Carnegie Mellon University, USA.

Zhang, Y., D. Chen, Y. Choi, L. Chen and S.B. Ko, 2010. A high performance ECC hardware implementation with instruction-level parallelism over GF(2^{163}). Microprocessors Microsyst., 34: 228-236.