Cost-Effective Implementations of *GF (p)* Elliptic Curve Cryptography Computations

Hakim Khali, MIEEE and Ahcene Farah, SIEEE

Faculty of Computer Science & Computer Engineering
Ajman University of Science & Technology, PO.BOX 346, Ajman, UAE

Summary
This research paper aims at analyzing the impact of exploiting the parallelism available in two common Elliptic Curve Cryptography (ECC) projective forms on speed and cost factors, assuming point-multiplication is implemented using the *m*-ary algorithm instead of the popular binary algorithm. Point-multiplication is implemented using scalable multipliers in order to replicate the design for varying-size security keys. Simulation results are shown for each projective form and a cross comparison is also performed as well.

***Key words*:** ECC, projective forms, modulo multipliers.

1. Introduction

ECC was first proposed in 1985 by N. Koblitz [1] and V. Miller [2]. Since then, a considerable amount of research has been performed on secure and efficient ECC implementations. One of the most important advantages of ECC over conventional public-key schemes is that much smaller key sizes are required to achieve the same security level [3]. As an example, it is reported in [3] that 160 bits ECC is equivalent to 1024 bits RSA [4]. Smaller key sizes make ECC suitable for embedded systems and wireless applications, which require combining performance with low-power hardware and smaller security certificates. Several ECC implementations have been reported in the literature at the software level and hardware level as well [5-13]. The performance of an ECC cryptosystem is mostly determined by an efficient implementation of its arithmetic over a Galois Field $GF(p)$ or $GF(2^m)$. ECC arithmetic is applied on points located on selected elliptic curves and includes point-addition, point-doubling and point-multiplication. When elliptic curve (EC) points are expressed in affine coordinates, ECC arithmetic includes multiplications and divisions (inversions), which are highly time-consuming. In order to eliminate these divisions (inversions), EC points are expressed in projective coordinates, where scalar multiplication becomes the operation to optimize in order to reach the targeted cost/performance constraints. In our research paper, we propose to analyze the impact of exploiting the parallelism available in two common projective forms on speed and cost factors, assuming point-multiplication is implemented using the *m*-ary algorithm instead of the popular binary algorithm. We target scalable multipliers in order to replicate the design for varying-size security keys. The remainder of this paper is organized as follows: Section 2 presents a basic background about ECC. Section 3 presents ECC point-multiplication algorithms and related sequential and parallel implementation of projective forms. Section 4 presents the architecture model and related cost/speed estimators using an existing scalable multiplier. Section 5 presents simulation results and section 6 concludes the paper.

2. ELLIPTIC CURVES OVER *GF (P)*

2.1 General Background on ECC Arithmetic

In this section, we will refer to [3] for a description of ECC arithmetic. The elliptic curve arithmetic is defined over Galois field $GF(p)$ where p is a prime number greater than 3 ($p > 3$). All arithmetic operations are modulo p. The elliptic curve equation E over $GF(p)$ is given by:
$y^2 = x^3 + ax + b$; where $p > 3$, $4a^3 + 27b^2 \neq 0$, and $x, y, a, b \in GF(p)$. There is also a single element named the *point at infinity* or the *zero point* denoted O, which serves as the additive identity. For any point $P(x, y) \in E$, we have: $P + O = P$.

Point addition and Point Doubling

Additions in GF(p) are controlled by the following rules:

$O = -O$
$P(x, y) + O = P(x, y)$
$P(x, y) + P(x, -y) = O$

The addition of two different points on the elliptic curve is computed as shown below:

$P(x_1, y_1) + P(x_2, y_2) = P(x_3, y_3)$; where $x_1 \neq x_2$
$\lambda = (y_2 - y_1)/(x_2 - x_1)$
$x_3 = \lambda^2 - x_1 - x_2$
$y_3 = \lambda(x_1 - x_3) - y_1$

The addition of a point to itself (point doubling) on the elliptic curve is computed as shown below:

$P(x_1, y_1) + P(x_1, y_1) = P(x_3, y_3)$;
$\lambda = (3(x_1)^2 + a) /(2y_1)$
$x_3 = \lambda^2 - 2x_1$
$y_3 = \lambda(x_1 - x_3) - y_1$

Point Multiplication

Multiplication of a point $P(x, y) \in E$ by a scalar k over GF(p) is defined as a series of point-additions given by: $Q = [k]P = \underbrace{P + P + \ldots + P}_{k \text{ times}}$. As stated in [3], point multiplication in ECC is a special case of the general problem of exponentiations in Abelians groups and benefits from all the techniques available for the shortest addition chain problem for integers. These techniques will be discussed in details in subsequent sections. Table 1 shows the cost of point addition and point doubling operations in affine coordinates.

Table 1. Cost of ECC operations in affine coordinates

Operations	Point addition	Point doubling
Additions (A)	6	4
Multiplications (M)	3	4
Inversions (I)	1	1
Total	6A + 3M+1I	4A + 4M+1I

If we neglect the costs of multiplications by small constants, table 1 shows that the inversion is the most expensive operation, followed by multiplications. To reduce the cost of ECC operations, projective coordinates are used to eliminate the need of field inversions. This aspect will be analyzed in next sections.

2.2 ECC Encryption and Decryption Processes

As stated in [14], several approaches to encryption/decryption using elliptic curves are analyzed in the literature. A basic encryption process requires that a plaintext message m is encoded as a point P_m on the elliptic curve. The corresponding ciphertext C_m consists of a pair of points given by: $C_m = \{kG, P_m + kP_B\}$, where k is a random positive integer generated by the source A, G the base point and $P_B = n_B G$ the point representing the public key of the destination B and n_B is the private key of the destination B. The decryption process is given by:

$C_m\{P_m + kP_B\} - n_B(kG) = P_m + kn_B G - n_B kG = P_m$.

Most of the processing time consumed by the encryption/decryption processes is consumed by the computations related to kG and kP_B . Therefore, increasing the encryption/decryption rates requires to speed-up the computations related to point-multiplication.

2.3 ECC Arithmetic Using Projective Coordinates

Group operations in affine coordinates involve finite field inversion, which is a very costly operation, particularly over prime fields [15]. These inversions can be avoided by using various coordinate systems. The most common ones are projective, Jacobian, modified Jacobian, and Chudnovsky-Jacobian [15]. In this paper, we will use and compare two projective forms as presented in table 2.

Table 2. Examples of Projective Forms

Projective form	**Projective coordinates**	**Equivalent affine coordinates**
Form 1	$P(X, Y, Z)$	$P(X/Z^2, Y/Z^3)$
Form 2	$P(X, Y, Z)$	$P(X/Z, Y/Z)$

The cost of converting from affine to projective coordinates is trivial. However, conversion in other direction costs 1I + 4M (projective form 1) and 1I + 2M (projective form 2). Tables 3 and 4 show point-adding and point-doubling operations, using projective coordinates. We will assume that the cost of a squaring operation is equivalent to a multiplication.

Table 3A: Point-adding Using Projective Form 1

Operations	Cost
$\lambda_1 = x_1 z_2^2$, $\lambda_2 = x_2 z_1^2$	4M
$\lambda_3 = \lambda_1 - \lambda_2$, $\lambda_4 = y_1 z_2^3$	2M + 1A
$\lambda_5 = y_2 z_1^3$, $\lambda_6 = \lambda_4 - \lambda_5$	2M + 1A
$\lambda_7 = \lambda_1 + \lambda_2$, $\lambda_8 = \lambda_4 + \lambda_5$	2A
$z_3 = z_1 z_2 \lambda_3$, $x_3 = \lambda_6^2 - \lambda_7 \lambda_3^2$	5M + 1A
$\lambda_9 = \lambda_7 \lambda_3^2 - 2x_3$, $y_3 = (\lambda_9 \lambda_6 - \lambda_8 \lambda_3^3)/2$	3M + 2A
Total	16M + 7A

Table 3B: Point-adding Using Projective Form 2

Operations	Cost
$\lambda_1 = x_1 z_2$, $\lambda_2 = x_2 z_1$	2M
$\lambda_3 = \lambda_2 - \lambda_1$, $\lambda_4 = y_1 z_2$	1M + 1A
$\lambda_5 = y_2 z_1$, $\lambda_6 = \lambda_5 - \lambda_4$	1M + 1A
$\lambda_7 = \lambda_1 + \lambda_2$, $\lambda_8 = \lambda_6^2 z_1 z_2 - \lambda_3^2 \lambda_7$	5M + 2A
$z_3 = z_1 z_2 \lambda_3^3$, $x_3 = \lambda_8 \lambda_3$	3M
$\lambda_9 = \lambda_3^2 x_1 z_2 - \lambda_8$, $y_3 = \lambda_9 \lambda_6 - \lambda_3^3 y_1 z_2$	3M 2A
Total	15 M + 6A

Table 4A: Point-doubling Using Projective Form 1

Operations	Cost
$\lambda_1 = 3X_1^2 + aZ_1^4$, $Z_3 = 2Y_1Z_1$	5M + 1A
$\lambda_2 = 4X_1 Y_1^2$, $X_3 = \lambda_1^2 - 2\lambda_2$	3M + 1A
$\lambda_3 = 8Y_1^4$, $\lambda_4 = \lambda_2 - 2X_3$	1M + 1A
$Y_3 = \lambda_1\lambda_4 - \lambda_3$	1M + 1A
Total	10 M + 4A

Table 4B: Point-doubling Using Projective Form 2

Operations	Cost
$\lambda_1 = 3X_1^2 + aZ_1^2$, $\lambda_2 = Y_1Z_1$	3M + 1A
$\lambda_3 = X_1Y_1\lambda_2$, $\lambda_4 = \lambda_1^2 - 8\lambda_3$	3M + 1A
$X_3 = 2\lambda_4\lambda_2$, $Y_3=\lambda_1(4\lambda_3-\lambda_4)-8(Y_1\lambda_2)^2$	4M + 2A
$Z_3 = 8\lambda_2^3$	2M
Total	12 M + 4A

Assuming the cost of an addition is neglected compared to the cost of a multiplication, tables 3 and 4 show that the cost and performance of point-adding and point-doubling operations are primarily affected by the cost and performance of multiplications. In the next section, we will present the main algorithms used to implement ECC multiplications.

3. ECC point-multiplication Implementations

As stated in [3], the binary method is the simplest and oldest efficient method for point multiplication. It is based on the binary expansion of k. The corresponding algorithm is presented below.

ALGORITHM: Point-Multiplication: Binary Method
INPUT: A point P and an integer k, such that $k = \sum_{j=0}^{\ell-1} k_j 2^j, k_j \in \{0,1\}$.
OUTPUT: $Q = [k]P$
1. $Q \leftarrow O$.
2. **For** $j = \ell - 1$ **TO** 0 **STEP** (-1) **DO**
3. $Q \leftarrow [2]Q$
4. **IF** $k_j = 1$ **THEN** $Q \leftarrow Q + P$
5. **RETURN** Q

If we assume that on average W, the number of ones in k, is equal to $W = \ell/2$, the binary method requires $(\ell - 1)$ point doublings and W point-additions. According to tables 3 & 4, the cost of one point-addition is higher than the cost of a point-doubling operation, in terms of number of multiplications and additions as well. This number can be reduced by using the *m*-ary method, as an alternative to the binary method. The *m*-ary method is based on the *m*-ary expansion of k. This method is given below.

ALGORITHM: Point-Multiplication: *m*-ary Method
INPUT: A point P and an integer k, such that $k = \sum_{j=0}^{d-1} k_j m^j, k_j \in \{0,1,\cdots,m-1\}$ and $m = 2^r, r \geq 1$.
OUTPUT: $Q = [k]P$

Pre-computation
1. $P_1 \leftarrow P$.
2. For i = 2 to m-1 do $P_i \leftarrow P_{i-1} + P$
3. $Q \leftarrow O$

Main loop
4. **For** $j = d - 1$ **TO** 0 **STEP** (-1) **DO**
3. $Q \leftarrow [m]Q$ -- r point-doubling operations
4. $Q \leftarrow Q + P_{k_j}$
5. **RETURN** Q

As shown above, the binary method is a special case of the *m*-ary method corresponding to $r = 1$. The number of doublings in the main loop is equal to $(d-1)r$ while the number of point additions is equal to d. Since $d = \lceil \ell / r \rceil$, the number of doublings in the *m*-ary method may be up to $(r-1)$ less than the $(\ell - 1)$ required by the binary method. The number of point additions is equal to d, which is on average $(r/2)$ less than the binary method. However, the effective performance depends on the choice of the projective form. This aspect will be analyzed in the next section. In this research paper, point-multiplication will be implemented using the *m*-ary algorithm. Other algorithms are described in details in [3].

3.1 Sequential Implementation

The time needed to implement point-multiplication using the *m*-ary method is mainly affected by the type of projective form and the parameters (ℓ, r). The sequential time needed to perform a point multiplication (main loop) using the *m*-ary method can be estimated as follows:

Projective form 1: $T_1(\ell, r) = \frac{\ell}{r}(r \times 10M + 16M)$. The speed-up $S_1(\ell, r)$ over the binary method $(r = 1)$ is given by:

$S_1(\ell, r) = \frac{T_1(\ell, 1)}{T_1(\ell, r)} = \frac{r \times 26M}{r \times 10M + 16M}$. The maximum value for $S_1(\ell, r)$ is given by: $S_1(\ell, r \to \infty) = 2.6$.

Projective form 2: $T_2(\ell, r) = \frac{\ell}{r}(r \times 12M + 15M)$. The speed-up $S_2(\ell, r)$ over the binary method is given by:

$S_2(\ell,r)=\frac{T_2(\ell,1)}{T_2(\ell,r)}=\frac{r\times 27M}{r\times 12M+15M}$. The maximum value for $S_2(\ell,r)$ is given by: $S_2(\ell,r\to\infty)=2.25$. These time estimates assume no pipelining is used and show an advantage of using the projective form 1 over the projective form 2 for a sequential implementation. They also show a performance gain over the binary method. A better performance can be achieved by exploiting the parallelism available in the computations related to projective forms.

3.2 Concurrent Implementation

A lot of amount of research is being conducted to find various approaches to accelerate ECC operations. Just to name a few, in [17], authors explore the use of Residue Number System (RNS) to speed-up multiplications. In [18], the author proposes to replace the intrinsic operation *double-and-add* by a new operation called *quad-and-add* expressed in radix-4 instead of the popular radix-2. In [19], authors propose to use Instruction-level parallelism (ILP) and multiple modular arithmetic logic to speed-up ECC operations, assuming the popular binary algorithm. In our research work, we want to exploit the parallelism available in projective forms at the operation level (fine grain parallelism) to accelerate ECC point multiplication, assuming the m-ary algorithm instead of the binary algorithm. Therefore, two levels of optimization are achieved: at the algorithmic level by using the *m*-ary algorithm and at the operation level by exploiting fine-grain parallelism. Fine-grain parallelism can be identified by exploring the dependency graph of the targeted operations. Figures 1 & 2 show the dependency graphs at the process level related to projective form 1 (PF1).

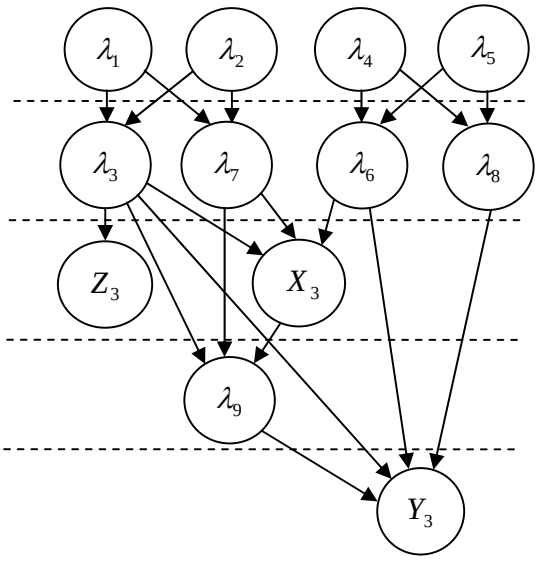

Fig. 1. Point-adding dependency graph (PF1)

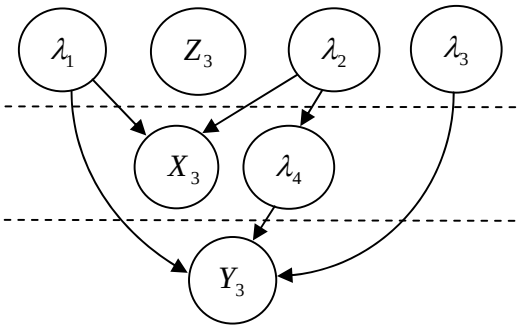

Fig. 2. Point-doubling dependency graph (PF 1)

The dependency graphs presented in figures 1 & 2 show that up to four processes can be run in parallel. To achieve the best performance, this amount of parallelism must be supported by adequate hardware resources. Figures 3 & 4 show partial dataflow graphs related to the processes that can be run in parallel. Graphs related to projective form 2 have not been included to avoid too many details. However they can be found in [16].

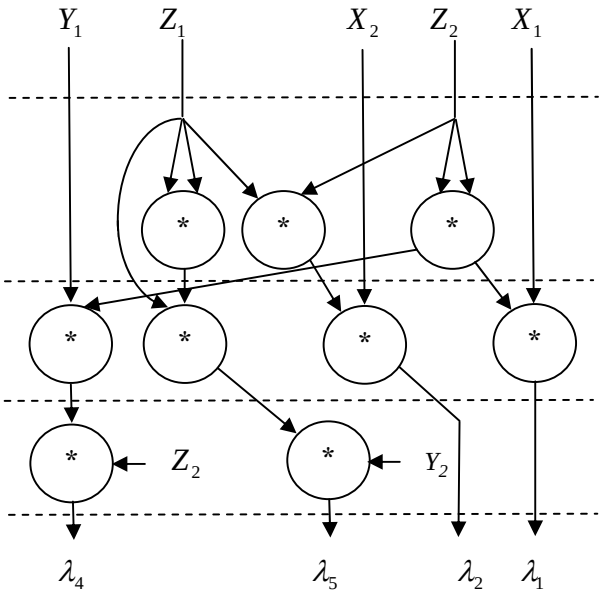

Fig. 3. Point-adding partial dataflow graph (PF 1)

Figures 3 and 4 show that point-adding and point-doubling operations related to projective form 1 can make use of up to 4 multipliers. The remaining processes are mainly based on additions or a less number of multiplications and therefore the same conclusion holds. It is then useless to allocate more than 4 multipliers to implement the projective form 1. The work presented in [16] also shows that the projective form 2 can make use of up to 4 multipliers. Allocating 4 multipliers to implement ECC operations expressed in projective forms will speed-up the

corresponding computations but will increase the cost as well. In a system design environment, cost and speed work together and cannot be dissociated. One of the main objectives of our research work is to analyze the impact of allocating several multipliers on speed and cost to reach cost-effective implementations.

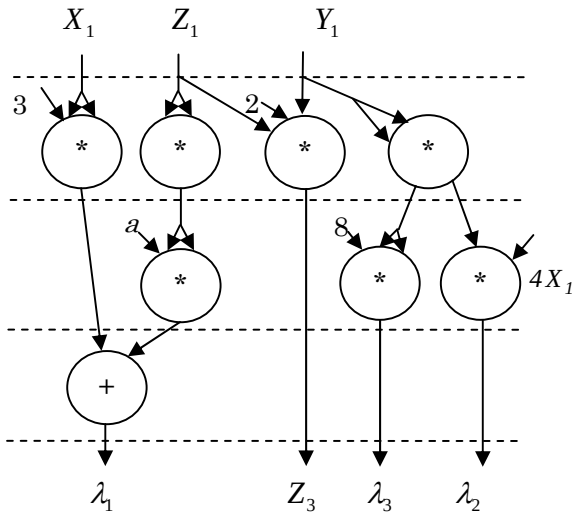

Fig. 4. Point-doubling partial dataflow graph (PF 1)

3.3 Allocation/Scheduling Tasks of ECC Operations

As stated in the previous section, cost and speed work together and cannot be dissociated during the design phase. A valid design is a design that meets cost and time constraints. To measure the quality of a design, several metrics can be used. In our research work, we will use an AT^2 metric where (A) is the area and (T) the time. Please note that time has been given more weight, while it is also common to give the same weight to area and time as well. We will assume the following:

- Area (A): it will be defined as the total area occupied by the number of multipliers allocated to compute ECC operations. The minimum number of multiplier is 1 and the maximum number of multipliers is 4 (see previous section). The cost of an adder will be neglected compared to the cost of one multiplier.
- Time (T): it will be defined as the number of multiplication cycles needed to compute ECC operations. Addition cycles will be neglected since addition requires less time compared to multiplication.

The area of a generic multiplier will be identified as A_{Mult} and the multiplication cycle will be identified as T_{Mult}. Scheduling the ECC operations on allocated resources can be achieved by using well-known scheduling algorithms such as ASAP or ALAP. Table 5 shows the results representing the number of multiplications control steps related to ECC operations depending on the number of allocated multipliers. Operations are scheduled as soon as operands and multipliers are available.

Table 5A. Number of multiplication control steps T_{Mult} required to schedule ECC operations (PF1)

#Multipliers N_{Mult}	Point-adding	Point-doubling
1	16	10
2	8	5
3	6	5
4	5	4

Table 5B. Number of multiplication control steps T_{Mult} required to schedule ECC operations (PF2)

#Multipliers N_{Mult}	Point-adding	Point-doubling
1	15	12
2	8	6
3	7	5
4	4	3

Using the results presented in table 5, the time needed to execute the *m*-ary multiplication algorithm can be estimated as follows:

$$T(N_{Mult}) = d(r \times T_{doubling}(N_{Mult}) + T_{adding}(N_{Mult})) \times T_{cycle}$$

, where T_{cycle} is the cycle time. The AT^2 metric can now be defined as follows:

$$AT^2(N_{Mult}) = A(N_{Mult}) \times T^2(N_{Mult})$$
$$= (N_{Mult} \times A_{Mult}) \times T^2(N_{Mult})$$

4. Architecture Model

Implementing the computations involved in projective forms using the *m*-ary multiplication algorithm requires the following basic units:

- A register file to store results and needed parameters.
- A set of multipliers (between 1 and 4) to implement multiplication operations.
- Adder/subs tractor.
- Controller.

The most critical issue is the choice of the multiplier type and its precision. Designing an optimized multiplier for a given precision may will lead to a good performance. However, this multiplier cannot be re-used for a higher precision with the same performance. Moreover, pipelining will result in very poor performance because of data dependencies related to the computations of point-adding and point-doubling operations. Therefore, designing ECC systems will favor flexibility to exploit the parallelism available in projective forms. One way to achieve this target is the use of scalable multipliers [20]. This flexibility allows the designer to reach a trade-off

between area and speed by reducing the bits per word size and the number of stages [20]. In our work, we will use scalable multipliers described in [20] to model the various hardware configurations and estimate area and speed as well. The same approach has been also used in [16]. However, it is worth to mention that this choice has been made to support our theoretical analysis with practical results but it is not the point of contribution of this work. As stated in [20], the area of a radix-8 scalable multiplier can be approximated as:

$$A_{Mult} \approx 92 * BPW * NS + 269 * NS - 9.42 * BPW - 35.5,$$

where BPW and NS represent the number of bits per word and the number of stages respectively. The number of clock cycles per multiplication NCC is estimated as:

$$NCC = \lceil n_{max} / (3 * NS) \rceil * (2 * NS + 1) + NW + 1, \quad IF \quad NW \le 2 * NS$$

$$NCC = \lceil n_{max} / (3 * NS) \rceil * (NW + 1) + 2 * NS \quad IF \quad NW > 2 * NS$$

, where $NW = \left\lceil \frac{\ell + 1}{BPW} \right\rceil$ and NS represent the number of words and the number of stages respectively. The number of multiplication control steps $MCS(N_{Mult})$ is given by table 5 depending on the number of multipliers allocated to the design. Therefore the total execution time needed to perform point-multiplication using the *m-ary* method is given:

$$T(N_{Mult}) = d(r \times MCS_{doubling}(N_{Mult}) + MCS_{adding}(N_{Mult})) \times NCC \times t_p,$$

Where t_p is the clock cycle time. It is worth to mention that t_p is also a function of BPW and NS.

5. Simulation Results

As stated in [20], the experimental values of the various design parameters have been obtained after the synthesis process using the Asic Design Kit (ADK) from Mentor Graphics and the AMI05_slow design technology. Three types of multipliers will be used in our experiments as shown in table 6.

Table 6. Multiplier types used in experiments

Type	Area (gates)	Multiplication cycle time (ms)	*NS*
1	895	0.5	1
2	9940	0.05	10
3	14965	0.03	15

Experiment 1

In this experiment, we want to evaluate the impact of the number of stages NS and the value of r on the total execution time, assuming one multiplier only. Figure 5 and figure 6 show the variation of the total execution time T as a function of r for various values of NS, assuming $\ell = 1024$ bits and $BPW = 8$.

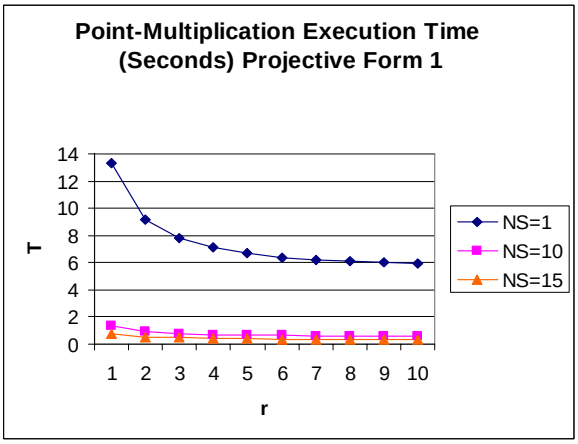

Fig. 5. Variation of execution time T as a function of r (PF 1)

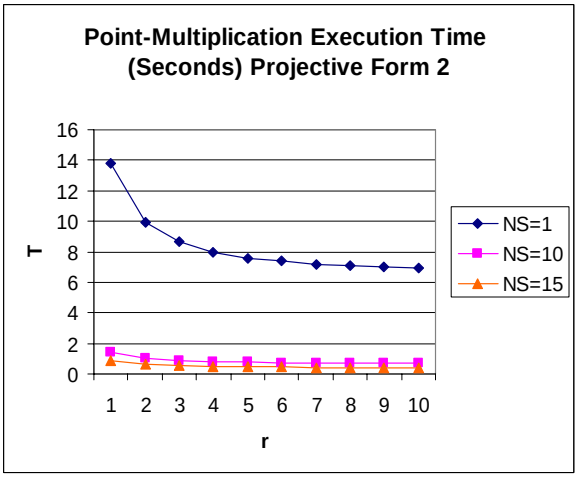

Fig. 6. Variation of execution time T as a function of r (PF 2)

Figures 5 and 6 show that for small values of r ($r \le 5$), increasing the value of NS considerably decreases the execution time T and achieves a substantial advantage over the binary method $(r = 1)$. However, when $r > 5$, the time gain is less affected. This is also true for $NS = 1$. These figures also show a slight advantage of projective form 1 over projective form 2 from a time point of view. Increasing the value of NS also allows going beyond the theoretical limits identified in section 4 when no pipelining is used. All these results suggest that a careful choice of

(r, NS) pair should be made at the system level to avoid unnecessary additional costs as a result of a high value of NS.

Experiment 2

In this experiment, we want to evaluate the impact of the number of multipliers and r on AT^2 metric. Figure 7 and figure 8 show the variation of AT^2 as a function of r, assuming $\ell = 1024$ bits, $BPW = 8$ and one to four type 2 multipliers.

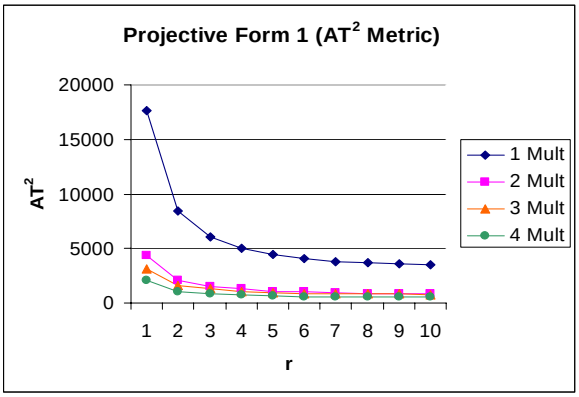

Fig. 7. Variation of AT^2 as a function of r (PF 1)

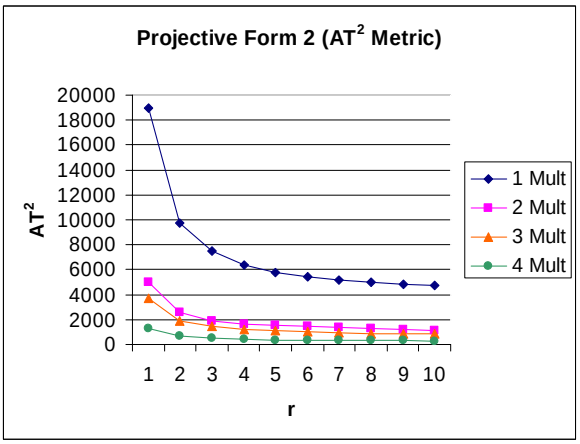

Fig. 8. Variation of AT^2 as a function of r (PF 2)

Figure 7 and 8 show that increasing the number of multipliers almost leads to the same AT^2 score for high values of r. This is due to the data dependencies available in the computations which prevent from using the hardware resources efficiently. Figure 9 compares the AT^2 metrics of projective form 1 and projective form 2 respectively.

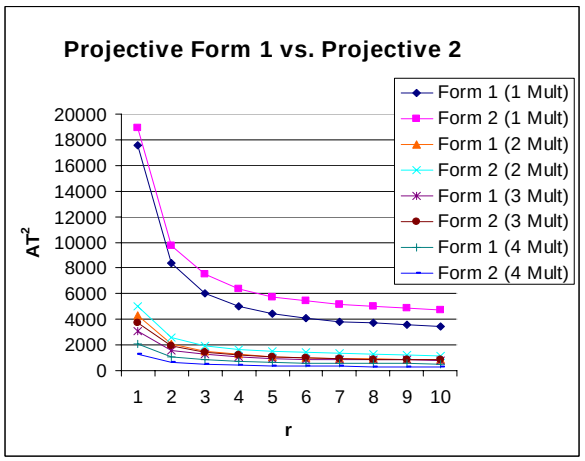

Fig. 9. Comparison of AT^2 metrics related to projective forms 1 an 2

Figure 9 shows that the AT^2 scores of projective form 1 is better than the scores of projective form 2 for the same number of multipliers, except for the case where $N_{Mult} = 4$. In this case, the utilization factor of the four multipliers related to projective form 2 is higher than the utilization factor of the four multipliers related to projective form 1, which leads to a faster implementation.

Experiment 3

In this experiment, we want to evaluate the efficiency of hardware configurations using more than one multiplier. This efficiency is computed as follows:

$E(N_{Mult}) = \frac{S(N_{Mult})}{N_{Mult}}$, where N_{Mult} is the number of multipliers allocated by the hardware configuration, and $S(N_{Mult})$ is the speed-up over a one-multiplier hardware configuration. Figures 10 and 11 show the resulting efficiencies.

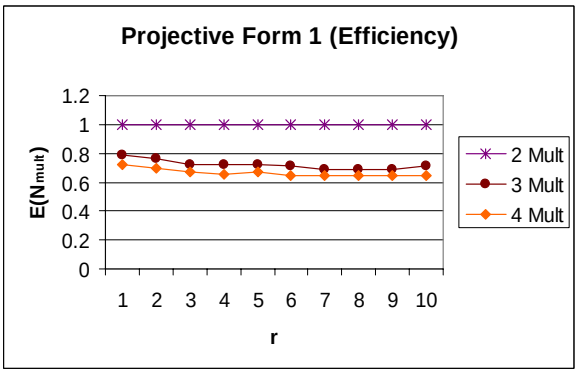

Fig. 10. Variation of $E(N_{Mult})$ metric related to projective form 1

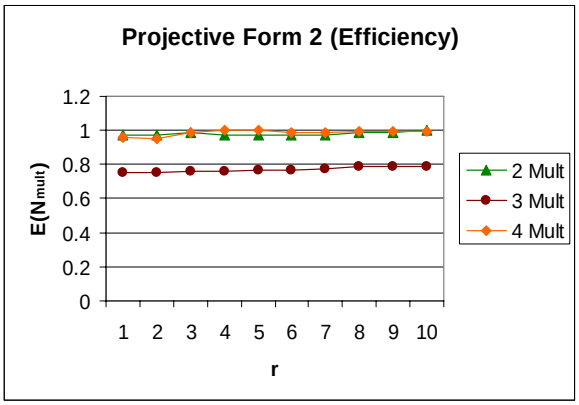

Fig. 11. Variation of $E(N_{Mult})$ metric related to projective form 2

Figure 10 shows that projective form 1 achieves its highest efficiency when $N_{Mult} = 2$, which indicates the best hardware utilization as a direct consequence of the data dependencies available in the computations, which prevent an efficient use of hardware resources. Figure 11 shows that projective form 2 achieves its highest efficiency when $N_{Mult} = 2$ or $N_{Mult} = 4$ due to its better hardware utilization.

6. Conclusion

In this research paper, we analyzed the impact of using the *m*-ary algorithm as an alternative to the binary algorithm to implement ECC multiplication operations in two common projective forms. This analysis took into account several aspects: time, AT^2 score, and efficiency. It clearly showed the superiority of the *m*-ary algorithm over the binary algorithm and speed-up factors can be doubled for high values of r. This performance can be enhanced by using the parallelism available in the computations related to projective forms. However, experiments showed that in some cases increasing the number of multipliers did not result in a substantial performance gain, as a direct consequence of the data dependencies available in various computations. This result is very important for low-power applications, where design constraints are usually severe. Results also showed that depending on the design parameter being analyzed one projective form may have the advantage over the other one. This suggests a careful choice of the projective form depending on design constraints.

References

[1] N. Koblitz, "Elliptic Curve Cryptosystems", *Math. Comp.*, 48, 1987, pp. 40-56.

[2] V. Miller, "Use of Elliptic Curves In Cryptography", In *Advances in Cryptology, CRYPTO-'85, Springer LNCS 218*, 1986, pp. 417-426.

[3] I. Blake, G. Seroussi, and N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press: New York, 1999.

[4] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of the ACM*, 21(2), 1978, pp. 129-126.

[5] S. Okada, N. Torii, K. Itoh, and M. Takenaka, "Implementation of Elliptic Curve Cryptographic Coprocessor over GF(2^m) on an FPGA", *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, pp. 25-40.

[6] G. Orlando, C. Paar, "A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m)", *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, LCS, Springer-Verlag, 1965, pp. 58-56.

[7] G. A. Orton, M. P. Roy, P. A. Scott, L. E. Peppard, S. E. Tavares, "VLSI implementation of public-key encryption Algorithms", *Advances in Cryptology -- CRYPTO '86*, Vol. 263 of Lecture Notes in Computer Science, Springer-Verlag, pp. 277-301,

[8] A. F. Tenca, and C. K. Koc, "A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm", *IEEE Transactions on Computers*, 52(9), 2003, pp. 1215-1221.

[9] A. Gutub, A. F. Tenca, and C. K. Koc, "Scalable VLSI architecture for GF(p) Montgomery Modular Inverse Computation", *IEEE Computer Society Annual Symposium On VLSI*, 2002, pp. 53-58.

[10] A. Gutub, A. F. Tenca, and C. K. Koc, "Scalable and Unified Hardware to Compute Montgomery Inverse in GF(p) and GF(2^n)", *Cryptographic Hardware and Embedded Systems - CHES 2002*, 2002, pp. 485-500.

[11] Royo, Moran, Lopez, "Design and implementation of a Coprocessor for cryptography applications", *European Design and Test Conference Proceedings*, 1997, pp. 213–217.

[12] S. B. Ors, L. Batina, B. Preneel, and J. Vandewalle, "Hardware Implementation of an Elliptic Curve Processor over GF(p)", *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP)*, 2003, pp. 433-443.

[13] D. Hankerson, J. L. Hernandez, and A. Menezes, "Software Implementation of Elliptic Curve Cryptography over Binary Fields", *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, LCS, Springer-Verlag,

2000.

[14] W. Stallings, Cryptography and Network Security: Principles and Practice, 3rd Ed., Prentice Hall, NJ, 2003.

[15] P. K. Mishra, “Pipelined Computations of Scalar Multiplication in Elliptic Curve Cryptosystems”, IEEE Trans. Comp. September 2006, pp. 1000-1010.

[16] A. A. Gutub, “Fast 160-Bits GF(P) Elliptic Curve Crypto Hardware of High-Radix Scalable Multipliers”, International Arab Journal, Vol. 3, No. 4, 2006, pp. 342-249.

[17] D. M. Schinianakis, A.P. Kakarountas, and T. Stouraitis, "A New Approach to Elliptic Curve Cryptography: an RNS Architecture", *Proceedings of IEEE MELECON*, 2006, pp. 1241-1245.

[18] S. Moon, " Elliptic Curve Scalar Point Multiplication Algorithm Using Radix-4 Booth's Algorithm", *ECTI Transactions on Computers and Information Theory*, 1(1), 2005, pp. 3-8.

[19] K. Sakiyama, E. De Mulder, B. Preneel, and I. Verbauwhede, "A Parallel Processing Hardware Architecture For Elliptic Curve Cryptosystems", *Proceedings of ICASSP*, Vol. 3, 2006, pp. 904-907.

[20] A. F. Tenca, G. Todorov, and C. K. Koc, "High-Radix Design of a Scalable Modular Multiplier", *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems, 2001*, pp. 185-201.

Dr. Hakim khali is an Assistant Professor of the Faculty of Computer Science & Computer Engineering at Ajman University of Science & Technology. He got his B.Sc in Computer Engineering from I.N.I (Algeria) in1989 and his M.Sc.A and PhD in 1993 and 2000 respectively from Ecole Poytechnique of Montreal (Canada). His research interests are Hardware-Software Codesign, VLSI architectures, and FPGA-based designs for Neural Networks and Cryptography. Before Joining Ajman University, he worked as a System Designer for Mirotech Microsystems on reconfigurable computing systems. Dr. Khali is an IEEE Member.

Prof. Ahcene Farah received the Electronics Engineer degree (1977) and Information Processing Master (1983) degree from the “Ecole Nationale Polytechnique” (ENP), Algiers, and Es-Science French State Doctorate degree (equivalent to PhD in computer engineering, 1989) from the National Polytechnic Institute of Lorraine, France (Institut National Polytechnique de Lorraine, INPL). He worked as teacher assistant from 1979 to 1984 at ENP. Between 1984 and 1989, he was Es-science Doctorate candidate at the INPL, and researcher at the Research Center in Automatics of Nancy. He was Associate-Professor (1990 –1998), then Professor (1998-1999) at the ENP, Algiers. September 1999– present, he is Professor at the faculty of Computer science and Computer Engineering, Ajman University, UAE. His research interests include**:** Computer and Network Information Security, Soft Computing (Neural Networks & Fuzzy Logic), Network Reliability, and Computer Forensic. He published more than 37 papers in Scientific Journals and Proceedings of International Conferences. He supervised 4 PhDs and 4 Masters of Science that have been completed and defended by the candidates. Prof. A. Farah is an IEEE senior Member.