Journal of Information Assurance and Security
ISSN 1554-1010 Volume 6 (2011) pp. 167-175


# Enhanced FPGA Implementations for Doubling Oriented and Jacobi- Quartics Elliptic Curves Cryptography

**Lo'ai Tawalbeh and Qasem Abu Al-Haija'**

Computer Engineering Department, Jordan University of Science and Technology,
P. O. Box 3030, Irbid, 22100, Jordan
*tawalbeh@just.edu.jo, Eng_Qasem1982@yahoo.com*

***Abstract*: The rapid advances in communication networks impose the fact of transferring big amounts of information worldwide every second. Considerable part of this information might need protection against fraud, modification and different types of intrusion. Both, the symmetric and asymmetric encryption algorithms provide different types of security services to protect sensitive information. Lately, the National Institute of Standards and Technology issued Federal Information Processing Standards to direct the researcher's attention to the asymmetric cryptographic algorithms based on elliptic curves. In this paper, algorithms and design issues related to two new curves: Doubling Oriented, and Jacobi-Quartics, are proposed and analyzed. Then, different implementation approaches are studied and applied for the different curves. Experimental results are provided to show the enhancements on the execution delay and the total design area of the proposed FPGA realizations.**

***Keywords*:** about Information Security, Enhanced Realizations, Doubling Oriented curves, Jacobi-Quartics curves, and Execution Delay**.**

## I. Introduction

The everyday developing communication networks is responsible for moving big amount of data between their infinitely many users. There might be hackers trying to hack certain sensitive data, which makes protecting these data a major concern of every single user of these networks. The protection process must be comprehensive and involve preventing different types of passive and active attacks, such as sniffing, spying, and identity fraud.

Using the private-key encryption algorithms, such as the TwoFish, RC4, Tripple DES (3DES), and Advanced Encryption Standard (AES) and many others [7,8], provides the required protection and has many advantages, that include the low realization complexity in hardware and software. On the other side, the problem of key distribution between the communicating parties is critical and considered a main limitation for the use of this category of algorithms. One solution to this problem is to use the public-key encryption algorithms which have, on the other hand, higher complexity realizations due to the vast mathematical operations involved.

In the last few years, the National Institute of Standards and Technology (NIST) issued many Federal Information Processing Standards (FIPS) to direct the researcher's attention to the asymmetric cryptographic algorithms based on elliptic curves. The FIPS 186-3 (ECDSA-June 2009) [1] recommends the use of these special curves in Digital Signature Algorithm to protect data in Federal Agencies. Also, the NIST Special Publication (SP) 800-56A (ECDH-March 2007) [2] recommends the Diffie-Hellman key exchange protocol that is based on these curves for government sensitive information exchange.

The above recommendations are based on many scientific research facts, and mainly the fact that these curves, once used in cryptography, can protect information at TOP SECRET level with relatively small key size when compared to other algorithms in this field [3]. Moreover, and to emphasize the importance of this new cryptography field, and to encourage the researches to propose hardware and software realizations for security algorithms based on these curves, the National Security Agency (NSA)issued guidelines to implement such algorithms, for example: Suite B Implementer's Guide to SP 800-56A [4]. From all the above, we can understand the motivation behind this research.

In this paper, we will study two new types of elliptic curves [5], the Doubling Oriented and Jacobi-Quartics curves [6]. We must mention that the original message to be encrypted, is first represented as a point on the selected curve using either the Affine coordinates system, or the projective (standard) coordinates system [5,8]. Then, there are mathematical operations performed on the curve points that include: Addition, Doubling, and Point Multiplication by a constant over a finite field (GF (p) or GF (2^n)). The algorithms that perform these mathematical operations have different complexity levels [10, 11, and 14].

In this paper, we will present these different algorithms that perform these point operations over the above mentioned two

curves. Moreover, more than one design approach are presented, and applied to get the designs that realize these algorithms. Adding to that many improving performance-related issues will be discussed and analyzed.

The next section presents the practical aspects of using the elliptic curves in cryptographic functions. In Section 3, the Doubling Oriented Doche-Icart-Kohel curves related point operations algorithms are derived and presented. Section 4 presents the algorithms to compute point operations for the Jacobi-Quartics curves. Section 5 presents three different implementation approaches, and examples of related work in the literature that adopted them. The experimental results are obtained and compared in Section 6, followed by conclusions in Section 7.

## II. Cryptographic Functions of Elliptic Curves

This Section presents the practical use of the mathematical elliptic curves in cryptographic functions, so the reader can get more clear idea on the application of these curves to provide information security. Furthermore, we will demonstrate in general how to use these curves in the encryption and decryption processes. Also, we will present the ElGamal encryption Algorithm in its integer version, and obtain the elliptic curve version of this algorithm, and demonstrate how this conversion between the two versions happened which can generally applied to any cryptographic algorithm.

We end this Section, by presenting a general formula of the Digital Signature Algorithm derived for the elliptic curves. We can find in the literature a related research that proposes cryptographic operations based on the elliptic curves. For example, an identity-based Proxy Signature Scheme based on the ECDSA is proposed in [7], and a scheme for key sharing based on elliptic curves is proposed in [18].

### A. *Encryption Using the Elliptic Curves*

For clarification purposes, we will introduce simple techniques to show the elliptic curves encryption/decryption process [8, 19]. Let's make the assumption that the points on the curve are reduced *mod* a certain prime integer called *p*. This means that all the mathematical operations on these points are carried out and reduced to keep the result in the range between {0 to p-1}. Before the encryption process starts, there is a key setup operation that must be performed as follows:

1. The users select a certain curve, (let's denote it by C).
2. The communicating users choose a base point (x, y) randomly that must satisfy the equation of the selected curve C.
3. The original message to be encrypted (denoted by the plaintext) is represented by a point $(x_m, y_m)$ that lies on the curve C.
4. Each user selects a private key, which is an integer, for example, for a certain user U, the private key will be $k^U$.
5. Each user computes his/her public key which would be his integer private key multiplied by the base point, that's it: $P_{PubKey} = k_U(x, y)$. We should mention that the resulting public key is again a point on the curve C.

Now, and after the key setup operation completed, the following steps demonstrate the encryption process:

1. For any one to encrypt and send the message point $(x_m, y_m)$ to a user U, he/she needs to choose a random integer N.
2. Then, generate the ciphertext which consists of two points on the curve: $Cm = \{N(x, y), (x_m, y_m) + NP_{PubKey}\}$.
3. There is no way to find out the plaintext from the ciphertext except by knowing the private key of the user U which is $(k_U)$.

To decrypt the ciphertext $(C_m)$ at the receiver side, the following steps are used which demonstrate the decryption process:

1. The point N(x, y) in the ciphertext $(C_m)$ is multiplied by the private key of the user U, to get the point: $k_U (N(x, y))$.
2. Then, this point is subtracted from the second point that constructs $C_m$, the result will be the plaintext point $(x_m, y_m)$.
3. The above two steps are summarized by:

$$((x_m, y_m) + N (P_{PubKey}) - k_U (N(x, y)) =$$
$$(x_m, y_m) + N (k_U(x, y)) - k_U (N(x, y)) = (x_m, y_m)$$

### B. *Elliptic Curve Version of ElGamal Encryption Algorithm*

The ElGamal cryptographic algorithm is based on the discrete logarithm problem. It is considered an efficient and secure public key algorithm that is used in many security applications [8]. Before we describe the elliptic curve version, we briefly mention the procedure of the integer (normal) version. The three major processes in ElGamal system are [8]:

1) Key Generation
2) The Encryption
3) The Decryption

If we assumed that Alice wants to send a message (m) to a certain receiver Bob, then the receiver (Bob) must generate his keys according to the following steps [8]:

1) Chooses a large prime (p).
2) Chooses an integer α mod p.
3) Chooses a secret integer a, and computes β, where:
$$\beta \equiv \alpha^{a} \pmod p.$$
4) The public key components are α, β, and p
5) The private key component is a (secret).

***Encryption:***

For Alice to encrypt the message (m), she has to choose a random integer i and computes:

$$x_1 \equiv \alpha^{i} \bmod p, \text{ and}$$
$$x_2 \equiv m\beta^{i} \bmod p$$

Then Alice sends the pair $(x_1, x_2)$ to Bob for decryption.

***Decryption*:**

After Bob receives the pair $(x_1, x_2)$, he decrypts it to get the message (m) as follows:

$$m \equiv x_2 \, x_1^{-a} \pmod p.$$

Now, after this brief explanation, we describe the ElGamal Elliptic Curve version.

***Key generation in the Elliptic Curve Version:***

Key generation process done by Bob is summarized by [8]:

1. Bob chooses an elliptic curve (C) defined over GF (p), where p is a large prime.
2. Bob chooses a POINT (α) on the selected curve (C).
3. Bob chooses a secret integer (a), and computes β = aα.
4. The public key components are the points α and β.
5. The private key component is a (kept secret).

***Encryption in the Elliptic Curve Version:***

Now, for Alice to encrypt here message, she has to do the following:

1. Alice has to represent her message as a point on the elliptic curve (C).
2. Then she chooses a random integer i.
3. And computes:
$$x_1 = i.\ \alpha, \text{ and}$$
$$x_2 = m + i.\ \beta$$

This pair $(x_1, x_2)$ is sent to Bob for decryption.

***Decryption in the Elliptic Curve Version:***

In order for Bob to get the message point, he computes:

$$m = x_2 - a.x_1$$

To prove that the above system works, we substitute the values of $x_1$ and $x_2$ in the above equation to get:

$$m = m + i.\ \beta - a.\ i.\ \alpha$$

But we know that the relation between α and β is: β = a. α.

By substituting the value of β, we get the message:

$$m = m + i.\ a.\ \alpha - a.\ i.\ \alpha = m$$

In general, to obtain an Elliptic curve version of a certain cryptographic algorithm, we have to replace certain mathematical operations in the original algorithm with other operations suitable for the points on the curve. These modifications are summarized in the following Table:

| **Operation in Original Alg.** | **Elliptic Curve Version** |
|---|---|
| Integer multiplication | Point addition |
| Integer exponentiation | Scalar point multiplication |

Table 1: Obtaining Elliptic Curve Version of a Cryptographic Algorithm.

From Table 1, we can convert any algorithm to the elliptic curve domain by replacing the integer multiplication by point addition, and the exponentiation is replaced by the Scalar point multiplication. These modifications were applied to the ElGamal algorithm to obtain the elliptic curve version as can be seen from the above explanation.

### *C. Digital Signature Algorithm Based on EC*

Due to many identity fraud accidents happened over the Internet and through non-secure transactions, the process of digitally sign the documents becoming more and more important. There are many algorithms to perform digital signature. Among the most secure algorithms is the Elliptic Curve Digital Signature Algorithm (ECDSA) [1, 8].

The ECDSA involves the following steps:
1. Key Generation
2. Signature Generation
3. Signature Verification

Each of these major steps is described in more details below.

***ECDSA Key Generation***

Each user of the digital signature scheme does the following steps to generate a key:

1. The user selects a curve (call it C), where the points on this curve are reduced mod a prime p (the curve is defined over the prime Galois finite field). We must mention that the number of points on the curve C should be divisible by a large prime n.
2. The user selects a point P = (x, y) of order n.
3. The user selects a random integer (i) such that $1 \le i \le n-1$.
4. The user multiplies the random integer (i) by the point P to get a new point on the curve: G = i.P.
5. The user's public key consists of (G, n, P, C). The user's private key is i.

***ECDSA Signature Generation***

After the key is generated, a message (m) can be signed as demonstrated by the following steps:

1. The user selects a random integer L such that $1 \le L \le n-1$.
2. The user multiplies the point P(x, y) by L to get a new point on the curve: LP= ($x_1$, $y_1$). Compute r = $x_1$ mod n. If r = zero then select new value for L as described in step 1 above.
3. The user computes $L^{-1}$ mod n.
4. The user compute the hash value of the original message (m) using a certain hash function to get h(m).
5. Then, compute s = $L^{-1}$ (h (m) + i. r) mod n. If s=0 go back to step 1.
6. The signature that will be used to sign the message (m) is both integers (s, r).

***ECDSA Signature Verification***

After the signature (s, r) is generated, it needs to be verified as described below:

1. The user needs to obtain an authentic copy of the public key (G, n, P, C).
2. Also, the values of r and s must be verified to be in the range {1, n-1}.
3. The user computes w = $s^{-1}$ mod n, and computes h (m) using the same hash function applied used in step 4 of the Signature Generation process.
4. Compute $v_1$= (h (m).w) mod n and $v_2$ = (r. w) mod n.
5. Then, compute $v_2G + v_1P = (x_2, y_2)$, and u = $x_2$ mod n.
6. If r = u, then the signature will be verified and accepted, otherwise, it will be rejected.

## III. Doubling Oriented Doche-Icart-Kohel EC.

The Doubling Oriented Doche-Icart-Kohel is defined by the equation [6]:

$$y^2 = x^3 + ax^2 + 16ax \quad modulo\ a\ prime\ p \qquad (1)$$

Where, the prime p is used to reduce the coefficients. The point doubling operation denotes the addition of a single point represented in the affine coordinates by P=($x_1$, $y_1$) on the curve to itself to get the result stored in the point P + P = ($x_3$, $y_3$), given that it is not equal to zero.
The algorithm that performs point doubling operation in general for points on any curve is given below [19, 24]:

$$(x_3, y_3) = 2(x_1, y_1)$$
$$x_3 = \omega^2 - 2x_1$$
$$y_3 = \omega(x_1 - x_3) - y_1$$
where: $\omega = \frac{dy}{dx}$, and y is the curve equation.

***Algorithm1,General point doubling operation for any curve***

The value of ω depends on the curve equation. For the Doubling Oriented curves, the value of ω is computed by taking the derivative of the curve Equation with respect to x to get:

$$2y.\frac{dy}{dx} = 3x^2 + 2ax + 16a \rightarrow \omega = \frac{dy}{dx} = \frac{(3x^2+2ax+16a)}{(2y)}$$

By substituting this value of ω in Algorithm 1 above, we get the customized steps to double a point P = ($x_1$, $y_1$) on the Doubling Oriented curves:

$$(x_3, y_3) = 2(x_1, y_1)$$
$$x_3 = {x_1}^4/(4 * {y_1}^2) - 8 * a/{y_1}^2 * {x_1}^2 + 64 * a^2/{y_1}^2$$
$$y_3 = {x_1}^6/(8 * {y_1}^3) + ((-a^2 + 40 * a)/(4 * {y_1}^3)) * {x_1}^4 + ((a * {y_1}^2 + (16 * a^3 - 640 * a^2))/(4 * {y_1}^3)) * {x_1}^2 + ((-4 * a^2 * {y_1}^2 - 512 * a^3)/{y_1}^3)$$

***Algorithm2, Customized point doubling operation for the Doubling Oriented curves.***

For implementation issues, many researchers prefer to use the projective coordinates system to represent the points on the curve. The reason behind that is to remove the division step in Algorithm 1, and replace it with many multiplication steps. The researchers realized that there is an urgent need to develop enhanced dedicated division algorithms to overcome the extra delay caused by this step.

*A. The Projection X/Z, $Y/Z^2$ for Doubling Oriented curves.*

There are many projections that can be used to replace the point (x, y) by the new point (X, Y, Z). Mainly, we will use the standard projection of (X/Z, $Y/Z^2$). Now, to derive the doubling algorithm for a certain point represented using the above projection of X/Z and $Y/Z^2$ on the Doubling Oriented curves, we need to replace P=(x, y) as follows: (x ➔ X/Z, y ➔ $Y/Z^2$), and the resulting ω will be:

$$\omega_{new} = \frac{\frac{3X^2}{Z^2} + \frac{2aX}{Z} + 16a}{\frac{2Y}{Z^2}} = \frac{3X^2 + 2aZX + 16aZ^2}{2Y}$$

Finally, and after substituting these new values in Algorithm 1, we get:

$$P = (X_1, Y_1, Z_1); 2P = (X_3, Y_3, Z_3)$$
$$(x, y) = (X/Z, Y/Z^2) \rightarrow (X, Y, Z)$$
$$T_1 = X_1^2 - 16a$$
$$T_2 = 2aX_1^2$$
$$Z_3 = 4Y_1^2$$
$$X_3 = T_1^2$$
$$T_3 = (Y_1 + T_1)^2 - Y_1^2 - X_3$$
$$Y_3 = T_3(X_3 + 64T_2 + a(2Y_1^2 - T_2))$$

***Algorithm3, Doubling of a point represented in projective coordinates on Doubling Oriented curve.***

If we assume that $Z_1$ = 1, then the cost to execute Algorithm 3 will be One multiplication and Five squaring operations and Seven additions.

For any value of Z1 ≠ 1, the resulting Algorithm will be:

$$\begin{aligned}&\mathrm{P} = (X_1, Y_1, Z_1); 2P = (X_3, Y_3, Z_3)\\&(x, y) = (X/Z, Y/Z^2) \rightarrow (X, Y, Z)\\&T_1 = X_1^2 - 16aZ_1\\&T_2 = 2aZ_1X_1^2\\&Z_3 = 4Y_1^2\\&X_3 = T_1^2\\&T_3 = (Y_1 + T_1)^2 - Y_1^2 - X_3\\&Y_3 = T_3 * (X_3 + 64T_2 + a(2Y_1^2 - T_2))\end{aligned}$$

***Algorithm4, Doubling of a point represented in projective coordinates on the Doubling Oriented curve for Z1 ≠ 1***

The cost in this case is two multiplications and five squaring operations and seven additions. In the next Section, we will follow the same methodology to derive the Algorithms for the Jacobi-Quartics curves.

## IV. Jacobi-Quartics Curves

The Jacobi-Quartics curve with coefficients reduced mod a prime (p) is defined by the equation [5, 6]:

$$y^2 = x^4 + 2ax^2 + 1 \quad modulo\ a\ prime\ p \qquad (2)$$

To derive equations to compute point doubling operation, we need to find the value of ω and substitute it in Algorithm 1. The value of ω depends on the curve equation, for the Jacobi-Quartics curves, the value of ω is computed by taking the derivative of the curve Equation with respect to x to get:

$$2y.\frac{dy}{dx} = 4x^3 + 4ax \rightarrow \omega = \frac{dy}{dx} = \frac{2(x^3+ax)}{(y)}$$

By substituting this value of ω in Algorithm 1 in the previous section above, we get the customized steps to double a point P=($x_1$, $y_1$) represented in the affine coordinates on the Jacobi-Quartics curve:

$$\begin{aligned}&(x_3, y_3) = 2(x_1, y_1)\\&x_3 = (2x_1 * y_1/(1 - (x_1^4))\\&y_3 = ((1 + x_1^4) * (y_1^2 + 2ax_1^2) + 2x_1^2(x_1^2 + x_1^2))/(1 - (x_1^4)^2\end{aligned}$$

***Algorithm5, Customized point doubling operation for the Jacobi-Quartics curves.***

In the next subsection, we will present the related algorithms to compute the point doubling operation for a point projected on the curve.

### *A. The Projection X/Z, Y/Z² for Jacobi-Quartics curves*

As mentioned earlier, there are many projections used to replace the point (x, y) with the new point (X, Y, Z). We will use the standard projection of (X/Z, Y/$Z^2$). Now, to derive the doubling algorithm for a certain point represented using the above projection of X/Z and Y/$Z^2$ on the Jacobi-Quartics curves, the value of x in the point p=(x, y) is replaced by X/Z, and the y value is replaced by Y/$Z^2$. These new values result in a new ω:

$$\omega_{new} = 2\frac{(\frac{X^3}{Z^3} + \frac{aX}{Z})}{\frac{Y}{Z^2}} = \frac{2(X^3 + aXZ^2)}{YZ}$$

By plugging these new values in Algorithm 1, we get the projective coordinate Doubling algorithm for the points over the Jacobi-Quartics curve:

$$\begin{aligned}&\mathrm{P} = (X_1, Y_1, Z_1); 2P = (X_3, Y_3, Z_3)\\&(x, y) = (X/Z, Y/Z^2) \rightarrow (X, Y, Z)\\&T_1 = (X_1 + Y_1)^2\\&T_2 = (X_1^2)^2\\&T_3 = Y_1^2 - T_2 - 2aX_1^2\\&X_3 = T_1 - Y_1^2 - X_1^2\\&Y_3 = (T_3 + T_2) * (Y_1^2 + 2aX_1^2) + 4T_2\\&Z_3 = T_3 - T_2\end{aligned}$$

***Algorithm6, Doubling of a point represented in projective coordinates on the Jacobi-Quartics curve.***

The assumptions made here are: $a^2 + c^2 = 1$, $Z_1 = 1$. The cost to execute Algorithm 6 with the above assumptions satisfied will be one multiplication and four squaring operations and nine additions.

For the general case, the value of $Z_1$ is not equal to one ($Z_1 \neq 1$), the Algorithm will be:

$$\begin{aligned}&\mathrm{P} = (X_1, Y_1, Z_1); 2P = (X_3, Y_3, Z_3)\\&(x, y) = (X/Z, Y/Z^2) \rightarrow (X, Y, Z)\\&T_1 = X_1^2 * Z_1^2\\&T_2 = (Y_1^2 + 2cT_1)^2\\&X_3 = (X_1Z_1 + Y_1)^2 - T_1 - Y_1^2\\&T_3 = X_3^2\\&Y_3 = T_2 - cT_3\\&Z_3 = Y_1^2 - 2aT_1 - 2X_1^4\end{aligned}$$

***Algorithm7, Doubling of a point represented in projective coordinates on the Jacobi-Quartics curve for $Z_1 \neq 1$.***

The cost in this case is one multiplication and seven squaring operations and seven additions.

## V. Implementation Approaches.

The direct hardware realization of Algorithm 2 to double a point (x, y) on the chosen curve involves many division operations. So, the researcher can directly think in three design and implementation approaches. The first approach is to propose an enhanced hardware to speed up the division (which has the longest execution delay among all other arithmetic operations). The second approach will be using an equivalent algorithm that has no division operation (replace it by multiplications), and this is the case of the projective coordinates (Algorithms 3,4 and 6,7). Finally, the third choice will be the Software Implementation.
In the next subsections, we will study the realizations that adopted each of these research directions, and how we can apply them to our new curves designs.

### *A. The First Approach: Enhanced Division Hardware.*

We can find much previous work that is related to this approach. Variety of division algorithms along with their implementations was proposed. We mention some of them in this subsection to explain the idea of this approach.

The researchers in [9] proposed an efficient realization to compute the modular division. The proposed design implements the Montgomery Inverse Algorithm which basically consists of computing two operations: the Montgomery product, and the "almost Montgomery inverse". There are many proposed Montgomery multipliers to compute the product efficiently [12, 13].

A modification of the work presented in [9] is proposed in [10]. A new correction step was introduced for the algorithm. This correction eliminated the multiplication (product) in the inversion process. The proposed hardware for the new modified algorithm benefit from the multi-bit shifting method, and has the main characteristic of scalability, so the user can change the operand size.

The salable designs consume less area and have better performance than the fully parallel ones, which make it a very efficient solution for the long precision division modular computations.

A different division algorithm that uses a counter method was proposed in [11]. The implementation of this new method simplified and reduced the division operation tremendously.

The researchers in [14] presented a systolic array implementation of Euclid's inversion based algorithm. The systolic array structure is built from a set of interconnected logic cells that are synchronized to perform a certain task. This design technique is very suitable for VLSI realizations, but on the other hand, it has a large area tradeoff.

The order of time complexity for these algorithms, it varies between O (1) and O ($n^2$), where the area complexity is in the range between O (n) and O ($n^3$), where n is the number of bits.

### *B. The Second Approach: Eliminating Division Operation.*

As mentioned at the beginning of this Section, the second alternative to get an enhanced realization of the elliptic curves algorithms is by getting rid of the division operation. In this Section, we will present some of the related work that adopted this approach.

In [21], the researchers presented generic architecture that can be adjusted to satisfy different area/performance requirements according to the size of the finite field of characteristic two.

The work proposed in [22] implements the equation of the standard Elliptic Curve given by the equation: $Y^2 = X^3 + aX + b$ over GF (p). The projective coordinates used are different ((X/Z, Y/Z) and ($X/Z^2$, $Y/Z^3$)). The proposed architecture is programmable in terms that you can choose the number of bits (size) of the prime (p), and it uses the idea of parallel multipliers to reduce the impact of using many multiplications instead of division operation.

The research in [23] proposed a high performance versatile architecture for scalar multiplication. The maximum field size is 255-bits. The design can efficiently process the fields of n=133 and n=193 bits.

A comprehensive survey about high-speed hardware implementations of Elliptic Curve Cryptography was presented in [24]. The authors summarized most of the related work to this research area. The classified the work based on many categories, which include: general architectures and arithmetic units and complete elliptic curve processors.

### *C. The Third Approach: Software Implementation.*

There are few realizations for elliptic curve operations in software. A good example is the work done [15]. A design that combined both hardware and software implementation was proposed in [16]. There are many reasons that direct the researches a way form this choice especially when it comes to high security applications. Software implementations are supported by operating systems (OS), which makes it vulnerable to all security threats associated with this OS, and as a result the application security will be compromised [17].

## VI. Experimental Results

In this research, we studied and analyzed each of the three possible approaches mentioned in the previous Section to obtain efficient realization of elliptic curve operations. Then, in this research we applied these approaches mentioned in the previous Section to propose three different implementations for the elliptic curves operations (mainly point doubling), for one of the two curves studied in this paper (Doubling Oriented Doche-Kohel-Icart).

### *A. The First Approach Results and Issues.*

The direct implementation of Algorithm 1 (points are presented in affine coordinates) must include enhanced division hardware. The researchers in this field proposed many efficient hardware algorithms to compute modular division. It is known that, if the total time to compute inversion (division) is less than the total time to perform the required multiplications replaced this division step, then it is worth it to perform the division to get an effecting hardware realization [10]. This equation depends on how many number of multiplications is needed to replace the division step, which basically depends on the curve used, and on the type of the projections used to obtain the projective coordinates representation of the point.

In our case, let's investigate the worst case scenario for the Doubling algorithms in projective coordinates for two curves. For the Doubling Oriented curves, Algorithm 4 shows that we need two multiplications and five squaring to perform point doubling. If we assume that the squaring time is equivalent to the multiplication time, we end up by seven multiplications. For the Jacobi-Quartics curves, Algorithm 7 shows the worst case scenario where one multiplication and Seven squaring operations are needed to perform the point doubling operation, and again if we made the assumption that the squaring time equals to a multiplication time, then

we end up by Eight multiplications. Form the above comparisons, we conclude that this approach will be efficient if we used a division algorithm that can perform the division operations required by Algorithm 2 (affine) in time less than the time required to perform seven multiplications for the Doubling Oriented curves. So, before we decide to use this approach in hardware realizations for the elliptic curve operations, we need to study the arithmetic algorithms carefully and their hardware implementation over the selected finite field. Efficient implementations for these algorithms can be found in [20].

*B. The Second Approach Results.*

In the previous subsection, we showed that in the worst case scenario ($Z1 \neq 1$), we need approximately 7 multiplication operations to compute the point doubling operation (projective coordinates) using to Algorithm 4 for the Doubling Oriented curves, while 8 multiplications are needed to perform the same operation (projective coordinates) using Algorithm 7 for the Jacobi-Quartics curves. According to the above, and to obtain optimum performance, we decided to impalement the doubling operation for the Doubling Oriented curves using Algorithm 4. The design was described in hardware description language (VHDL), and simulated using the Mentor Graphics simulation tool (ModelSim) [25] to verify the results. Finally, the FPGA target chip (xc5vlx110) was selected to synthesize the design using Xilinx Synthesize tool [26], to obtain area and delay results.

Table 2 shows the critical path delays (in nano-seconds) curves for different precisions starting form 16 to 256 bits.

| **Precision (bits)** | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| **Delay (Nano Sec)** | 13.2 | 13.8 | 14.1 | 14.3 | 14.4 |

Table 2: Critical Path Delay Results.

It can be seen from this Table that the minimum delay (clock period) is 13.2 nano sec which happens precision of 16 bits. The maximum occurred at 256 bits which is equal to 14.4 nano sec. This behavior is expected since the design complexity is a function of the operands size which is in terms determined by the finite field size.

The critical path delay presented in the Table will be used to compute the total time required to perform the Scalar Point multiplication operation, which is the main operation in any Elliptic Curve computations.

In general, the required time to compute one sequential multiplication is given by:

$$T = (cycles/bit) * n * clockperiod. \quad (3)$$

In our implementation (that uses Doubling Oriented curves), it takes one cycle to perform the operation on one bit, and so, the time required to compute one multiplication at operand precision of n = 512 bits is $T_{Dou\_Mul}$:

$$T_{Mul} = 1 * 512 * 14.7 * 10^{-9} = 7.53\mu sec \quad (4)$$

Also, we can compute the time required by point addition ($T_{Add}$) which requires 8 multiplications [6], and point doubling ($T_{Dbl}$), which requires 7 multiplications, at operand precision of n = 512 bit as follows:

$$T_{Add} = 8 * T_{Mul} = 60.24\mu sec \quad (5)$$

$$T_{Dbl} = 7 * T_{Mul} = 52.71\mu sec \quad (6)$$

The scalar point multiplication (multiplying a point on the curve (P) by a constant k to get kP, using Double-and-Add method [19, 24]. On average, for maximum precision of n-bits in a certain GF (p) field, the number of point doubling operations needed are n, and approximately n/2 point additions are needed [19, 22].

If the result needed to be converted back to the affine coordinate, we have to consider the time for one inversion operation. The time needed to compute modular for our design to be effecting more than the affine coordinates based design (first approach); we need to consider the WORST case which estimated the inversion time by 7 multiplication times. Based on the above approximations, we are able now to find the equation to compute the Scalar point multiplication time $T_{SM}$ to be:

$$T_{SM} = 0.5n * T_{Add} + n * T_{Dbl} + T_{inv} \quad (7)$$

$$= 0.5n * (8T_{mult}) + n * (7T_{mult}) + (7T_{mult})$$

which yields to:

$$T_{SM} = (11n * T_{mult} + 7T_{mult}) \quad (8)$$

By substituting the expression for $T_{mult}$ in the above Equation, we get:

$$T_{SM} = (11n+7) * T_{mult} = (11n+7) * n * \text{clock period}$$

$$T_{SM} = (11n^2 + 7n) * clockperiod \quad (9)$$

Lets take n=256 bits, then form Table 2, the time delay (clock period) is 14.4 nano sec. And we can compute the $T_{SM}$ for the Doubling curve as follows:

$$T_{SM} = [11(256)^2 + 7(256)] * 14.3\ nano\ sec = 2.61 \text{ milli sec}$$

By performing the same computations, the $T_{SM}$ at n =256 will be 10.4 ms. For comparison purposes, we compare our work with related FPGA implementations that used the same approach (second approach: projective coordinates). We can see that our proposed FPGA implementation computes the scalar point multiplication in 2.61 ms, and 10.4 ms at operands sizes 128 and 256, respectively. The proposed FPGA elliptic curve generator in [27] computes the point multiplication in about 3.84 ms at precision of 192 bits. Also, the proposed design in [28] computes the point multiplication in a relatively close amount of time which is about 3 ms. From the above discussion, we can notice that our FPGA implementation gives close results to related

implementations that uses different curves (basically the standard elliptic curve). This result proves that new elliptic curves can be used in cryptography such the Doubling oriented and gives the same performance. For the area analysis, Table 3 shows the obtained area results from the synthesis:

| **Precision (bits)** | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| **Area (No. of Gates)** | 2412 | 4204 | 7788 | 14956 | 29292 |

Table 3: Area Results.

We can notice form Table 3 that the minimum and maximum area occurred at when the precisions 16 and 256 bits, respectively, and as expected. It is also can be found from the Table the area equation to be:

$$AREA_{Dou} \approx 113.2 * n + 600 = O(n) gates \qquad (10)$$

We can conclude that the order of area complexity of our proposed design is O (n), which means that the area increases linearly with the operand size increment. This result is approximately similar to the area of many other designs that have O (n) area complexity [22, 27, and 28].

### *C. The Third Approach Results*

The third implementation approach is the software implementation. We implemented our design in software with the following personal computer specifications:

- Intel Core 2 Duo CPU T7500 @ 2.20 GHz.
- GB RAM.
- Windows 7 Operating System.

The implementation was programmed in C#.net. The results of these implementations were performing the point doubling operation in about 15 ms at operand size of 128 bits. This is much slower than the FPGA implementation proposed according to second approach in the previous subsection. Another reason that makes this approach is not preferable is the security vulnerabilities associated with the hosting operating system, which in turns affect the security of the implementation directly.

## VII. Conclusions

In this paper, algorithms to compute point doubling operation on two new curves: Doubling Oriented, and Jacobi-Quartics, is proposed and analyzed.

The algorithms perform the computations for points defined on affine and projective coordinates. Then, we studied different implementation approaches. Mainly, the first approach is the affine coordinate's implementation which requires the use of division operation, and the second approach is the projective coordinate's implementation that substitutes the division by many multiplication operations, and finally, the software implementation. Then, we investigated many examples from the literature that used these approaches, and we discussed many design issues to enhance the performance. Based on that, we found that the best approach in terms of efficiency is the second approach, and so, we applied it and used it to implement the point operations for the Doubling Oriented curve defined over GF (p).

To obtain the experimental results, the best case which is using the projective coordinates algorithm with the Doubling Oriented curves, was implemented in FPGAs. The target chip was selected to be xc5vlx110. The critical path delay synthesis results were used to compute the total time required to compute scalar point multiplication using Doubling Oriented curve with the projective coordinates is 2.61 ms at 128-bit operand precision, and 10.4 at 256 operand precision. This result is close to many other FPGA implementations.

We can conclude from this research, that we can use new elliptic curves in cryptography with new projective coordinates other than the Standard Curves (given by the equation $Y^2 = X^3 + aX + b$) and the ordinary coordinate of (X/Z, Y/Z). The Doubling Oriented elliptic curve FPGA implementation introduced here with the projection (X/Z, $Y/Z^2$) gave almost similar results to the implementations of the most common used curves and projective coordinates. The area results showed a complexity of O (n), and this is similar to many other designs, which supports our conclusion.

## Acknowledgments

Authors would like to thank Jordan University of Science and Technology, and the Ministry of High Education in Jordan (Scientific Research Support Fund) for supporting this research.

## References

[1]. U.S. Department of Commerce/National Institute of Standards and Technology (NIST), FIPS PUB 186-3: Elliptic Curve Digital Signature Algorithm, June 2009.

[2]. U.S. Department of Commerce/National Institute of Standards and Technology (NIST)-Special Publication (SP) 800-56A: Elliptic Curve Diffie-Hellman Key Exchange. Website: http://csrc.nist.gov, March 2007.

[3]. U.S. National Security Agency (NSA) - Suite B Cryptography. Website: http://www.nsa.gov/ia/, 2010.

[4]. U.S. National Security Agency (NSA) - Suite B Implementer's Guide to SP 800-56A. July 2009. Website: http://www.nsa.gov/ia/files/Suite_B_ImplementerG-13808.pdf.

[5]. N. Koblitz. "Elliptic curve cryptosystems", Mathematics on Computation, 48(177), pp. 203–209, 1987.

[6]. Explicit - Elliptic Curves Formulas, Website: http://hyperelliptic.org/EFD/g1p/index.html, 2010.

[7]. Ming-H. Chang et al."Enhance ECDSA to Identity-based Proxy Signature Scheme", Journal of Information Assurance and Security, 5(1) pp. 360-367, 2010.

[8]. W. Trappe and L. Washington. Introduction to Cryptography With Coding Theory, Prentice Hall, 2006.

[9]. E. Savas, and C. Koç. "The Montgomery modular inverse - Revisited", IEEE Trans. on Computers, 49(7): 763-766, July 2000.
[10]. Gutub, et al. "Scalable VLSI Architecture for GF (p) Montgomery Modular Inverse Computation". In Proceedings of the ISVLSI 2002 - IEEE Computer Society Annual Symposium on VLSI, Pittsburgh, Pennsylvania, April 2002.
[11]. Tenca and L. Tawalbeh, "An algorithm for unified modular division in (GF (p) and (GF (2^n)) suitable for cryptographic hardware", IEE Electronics Letters, 40 (5), pp. 304-306, March 2004.
[12]. A. Bernal and A. Guyot. "Design of a modular multiplier based on Montgomery's algorithm". In the Proceedings of the 13th Conference on Design of Circuits and Integrated Systems—DCIS'98, pp. 680–685, 1998.
[13]. S. Eldridge and C. Walter. "Hardware implementation of Montgomery's modular multiplication algorithm", IEEE Transactions on Computers, 42(6), 693–699, 1993.
[14]. J. Guo and C. Wang."Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in GF(2m)", IEEE Trans. on Computers, 47(10), pp. 1161-1167, October 1998.
[15]. Hankerson, Hernandez, and Menezes. "Software Implementation of Elliptic Curve Cryptography over Binary Fields". In The Proceedings of the Workshop on the Cryptographic Hardware and Embedded Systems-CHES, Massachusetts, August 2000.
[16]. S. Janssens and et al."Hardware/software co-design of an elliptic curve public-key cryptosystem". In The Proceedings of the IEEE Workshop on Signal Processing Systems, Belgium, Sep. 2001.
[17]. Stinson. Cryptography: Theory and Practice, CRC Press, Boca Raton, Florida, 1995.
[18]. J. Kar and B. Majhi. "A Secure Two-Party Identity-Based Key Exchange Protocol Based on Elliptic Curve Discrete Logarithm Problem", Journal of Information Assurance and Security, 5(1) pp. 473-483, 2010.
[19]. IEEE P1363, Standard specifications for public key cryptography, 1999.
[20]. Jean-Pierre Deschamps and et al., Hardware Implementation of Finite-Field Arithmetic, McGraw-Hill, NY, 2009.
[21]. M. Bednara and et al.."Reconfigurable Implementation of Elliptic Curve Crypto Algorithms". In the Proceedings of the International Parallel and Distributed Processing Symposium, Florida, April 2010.
[22]. L. Tawalbeh, et al., "Efficient FPGA Implementation of a Programmable Architecture for GF(p) Elliptic Curve Crypto Computations", Journal of Signal Processing Systems, 59(3), pp. 233-244. Springer, June 2010.
[23]. H. Eberle, N. Gura, S. Chang-Shantz. "A cryptographic processor for arbitrary elliptic curves over GF (2^m)". In the Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP), pp. 444–454, 2003.
[24]. G. de Dormale and Jean-J. Quisquater. "High-speed hardware implementations of Elliptic Curve Cryptography: A survey", Journal of Systems Architecture, 53(2-3), pp.72-84, February-March 2007.
[25]. Mentor Graphic Corporation, ModelSim Tutorial Software, Website:http://www.mentor.com, 2011.
[26]. Xilinx Company, Website:http://www.xilinx.com, 2011.
[27]. McIvor, M. McLoone, and J. McCanny. "An FPGA elliptic curve cryptographic accelerator over GF(p)". In the Proceedings of the Irish Signalsand Systems Conference (ISSC), pp. 589–594, 2004.
[28]. Orlando, C. Paar. "A scalable GF(p) elliptic curve processor architecture for programmable hardware". In the Proceedings of the Cryptographic Hardware and Embedded Systems (CHES), LNCS 2162, pp. 356–371, 2001.

## Author Biographies

**Lo'ai Tawalbeh** is an assistant professor of Computer Engineering at Jordan University of Science and Technology (JUST). He was born in a city north of Jordan called Irbid in 1977. He received his B.S. in Electrical and Computer Engineering from Jordan University of Science and Technology in June of 2000. Then he worked as a research and development engineer in a leading software company and as a Teaching Assistant in JUST before he joined the graduate program at Oregon State University (OSU) in September 2001. Dr. Lo'ai received his M.S. degree in Electrical and Computer engineering from Oregon State University under the direction of Dr. Alex Tenca in October 2002, and his Ph.D. under the direction of Dr. Cetin K. Koc ; in October 2004. Dr. Lo'ai research interests include network and computer security. Intrusion detection and computer forensics. Hardware implementations for cryptography, and cryptographic co-processor design using scalable modules, Elliptic Curve Cryptography, network security and embedded systems, FPGA design, VLSI design, computer architecture and finite field arithmetic algorithms. Dr. Lo'ai is a member of IEEE and IEEE Computer Society.
Hobbies: Swimming, racing and camping.

**Qasem Abu Al-Haija'** is computer engineer Senior Technical Instructor at Computer Technical Solutions (CTS), Also, Consultant and Researcher in the field of Cryptography and Information Security Security for Mohammad Alkhatib who is a PHD student under the Supervision of Prof. Azmi B. Jaafar, University Putra Malaysia, Department of Information System and Institute for Mathematical Research. He was born in a city north of Jordan called Irbid in 1982. He received his B.S. in Electrical and Computer Engineering from Jordanian Mu'tah University in February of 2005. Then he worked as a network engineer in a leading institute at KSA, and as a lecturer before he joined the graduate program at Jordan University of Science and Technology (JUST) in September 2007. Eng. Qasem received his M.S. degree in Computer engineering from Jordan University of Science and Technology under the direction of Dr. Lo'ai Tawalbeh in December 2009. Eng. Qasem research interests include Cryptography and Security, Computer Arithmetic and Finite Fields, Hardware implementations for cryptography, Wireless Sensor Networks, FPGA design, Elliptic Curve Cryptography, computer architecture, digital arithmetic algorithms. Hobbies: Reading-Writing, Swimming, Football.