HELSINKI UNIVERSITY OF TECHNOLOGY
Signal Processing Laboratory

Elliptic Curve Cryptography on FPGAs

Kimmo Järvinen
kimmo.jarvinen@hut.fi

June 9, 2003

Helsinki University of Technology
Signal Processing Laboratory
Otakaari 5A, 02150 Espoo, Finland

Contents

Version 1.0
This report was written using LaTeX and the figures were plotted with MathWorks Matlab® 6.5.

Abbrevations

AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
ANSI	American National Standards Institute
ASIC	Application Specific Integrated Circuit
CPU	Central Processing Unit
DL	Discrete Logarithm
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECP	Elliptic Curve Processor
ECIES	Elliptic Curve Integrated Encryption Scheme
FF	Finite Field
FIPS	Federal Information Processing Standards
FPGA	Field Programmable Gate Array
FPSLIC	Field Programmable System Level Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IF	Integer Factorization
ISO	International Standards Organization
LUT	Look-Up Table
MQV	Menezes-Qu-Vanstone signature scheme
NIST	National Institute of Standards and Technology
ONB	Optimal Normal Basis
PB	Polynomial Basis
RSA	Public-key cryptography algorithm invented by Rivest, Shamir and Adleman
SECG	Standards for Efficient Cryptography Group
SHA	Secure Hash Algorithm

Mathematical Notations

$GF(q)$	Galois field of order q
E	Elliptic curve
$\oplus$	Addition in Finite (Galois) field $GF(2)$, i.e. exclusive-or (xor)
$\bullet$	Finite (Galois) field multiplication
$\cdot$	Algebraic multiplication of polynomials
O_∞	Point at infinity, i.e. identity element of elliptic curve addition
$0x$	Hexadecimal notation
$0b$	Binary notation

Preface

This technical report was written as a part of the GO-SEC project in the spring 2003. The work was supervised by senior researcher Matti Tommiska and the author would like to thank him for his valuable comments. This report is also a preliminary work for the author's master's thesis of elliptic curve cryptography on FPGAs which will be written later this year.

Kimmo Järvinen
Otaniemi, June 9, 2003

1 Introduction

In 1985, Neil Koblitz [16] and Victor Miller [20] independently proposed the use of elliptic curves for cryptography. They did not invent a new cryptographic algorithm but they implemented existing public-key algorithms, like Diffie-Hellman, using elliptic curves [30]. Since 1985, there has been a lot of studies concerning elliptic curve cryptography (ECC) and in the past few years most of the big problems, that prevented ECC to really break into the main stream of cryptography, have been solved [27].

The use of elliptic curves in cryptography is very inviting for various reasons. The first and probably most important reason is that ECC offers better security with a shorter key length than any other public-key cryptography. For example, the level of security achieved with ECC using a 173-bit key is equivalent to conventional public-key cryptography (e.g. RSA) using a 1024-bit key [3]. Shorter key lengths also mean that less memory is needed in key storage which is in huge importance specially in applications having limited memory resources. Elliptic curve cryptosystems require less hardware resources than conventional public-key cryptosystems. For example, an implementation of 155-bit ECC using 11,000 transistors has been reported in [1]. This is a very small number compared to 50,000 transistors needed to implement 512-bit RSA [18]. ECC is also probably even more secure than RSA, as the largest RSA, that can be cracked succesfully, uses 512 bits and for ECC the number of bits is 97, respectively. In addition, the computer power used for cracking ECC is approximately twice the power used for RSA [18].

The mathematics used for ECC are considerably deeper and more difficult than mathematics used for conventional cryptography. This is in fact the main reason, why elliptic curves are so good for cryptographic purposes, but it also means that in order to implement ECC a lot more understanding of mathematics is required. A short introduction to mathematics behind elliptic curve cryptosystems is given in this report, but many implementations use very sophisticated methods to implement ECC and details of these methods are not considered here. However, this presentation should give a good overall picture of ECC and its implementation issues.

Recently, there has been a lot of studies concerning ECC in both software and hardware. This report concentrates on hardware implementations and specially on implementations on Field Programmable Gate Arrays (FPGAs). FPGAs are very attractive platforms for cryptographic implementations because they offer the speed of hardware combined with the flexibility of software. For low volume cryptography implementations FPGAs are also significantly cheaper alternatives than Application Specific Integrated Circuits (ASICs).

This report is organized as follows: in Section 2 basics of Galois (finite) fields are presented and in Section 3 mathematics of elliptic curves are discussed. In Section 4 principles of ECC and certain algorithms using ECC are presented. Implementation issues of ECC on FPGAs and certain implementations are presented in Section 5. Finally conclusions are made in Section 6.

2 Galois Fields

Galois fields are fields with a finite number of elements. They are named after their inventor Evariste Galois (1811-1832). Galois was a French mathematician who led an interesting, though short, life in the early 19th century. A biography of Galois can be found from Reference [4].

Galois fields are fields with a finite *field order* q which is also the number of elements in the field [11]. A Galois field of order q is here denoted as $GF(q)$. The order q of the field is always a prime p or a power of a prime p^m [11]. Galois fields $GF(p)$ are called *prime fields* and they are not considered here. In this report only Galois fields $GF(p^m)$ where $p = 2$ are considered. These fields are called *binary fields* and they are denoted as $GF(2^m)$.

Galois fields are commonly used for cryptography purposes but there are many applications using Galois fields also e.g. in the study of error-correcting codes. Field orders (field sizes) used for cryptography are usually huge (e.g. $m > 150$) in order to make cryptoanalysis harder. The security offered by the cryptosystem usually increases exponentially when m becomes larger. For cryptography applications m in $GF(2^m)$ should be a prime, in order to avoid a cryptoanalysis attack called Weil decent attack [7].

In Galois field $GF(2)$ there are two elements i.e. 0 and 1. Multiplication in $GF(2)$ yields for an and-operation and addition is performed with an exclusive-or-operation (xor). Thus, these calculations are very easy to implement in both hardware and software. This is actually the main reason why binary fields are so commonly used in cryptography.

When elliptic curve cryptography is implemented over Galois fields $GF(2^m)$, mainly two kinds of basis for the field are used. These bases are called *polynomial basis* (PB) and *(optimal) normal basis* (ONB). Polynomial basis, which is the traditional and more commonly used, is presented in Section 2.1 and normal basis is discussed in Section 2.2.

2.1 Polynomial Math over Finite Fields

Galois fields with a polynomial basis are most commonly used in elliptic curve cryptography and therefore they are presented here in detail. The other commonly used basis is called optimal normal basis (ONB), which will be later discussed in Section 2.2. Polynomial basis has proven to be faster and easier to implement than optimal normal basis and it is therefore usually prefered to ONB.

Galois field with a polynomial basis is generated with an *irreducible polynomial*[1] over $GF(2)$. Galois field of order N is generated with an irreducible polynomial $m(x)$ of degree N by setting $GF(2^N) : GF(2)[x]\,/\,m(x)$. Thus there are many representations of the same Galois field $GF(2^N)$ generated with different irreducible polynomials of degree N. Irreducible polynomials used for Galois fields

[1]A polynomial is said to be irreducible if it cannot be factored into nontrivial polynomials over the same field i.e. in this case over $GF(2)$.

in elliptic curve cryptography are usually of *trinomial* or *pentanomial* form. A trinomial polynomial has three non-zero terms, and for a pentanomial polynomial the number of non-zero terms is five.

Galois field $GF(2^N)$ with a polynomial basis has a basis $\{1, x, x^2, \ldots, x^{N-1}\}$ and an element of that Galois field can be expressed as a polynomial of degree $N-1$. Element $a(x)$ is expressed in polynomial form as follows:

$$a(x) = \sum_{i=0}^{N-1} a_i x^i, \tag{1}$$

where coefficients a_i are elements in $GF(2)$ i.e. 0 or 1. Thus N bits are needed to represent an element in $GF(2^N)$. The most significant bit (MSB) is the coefficient a_{N-1} and the least significant bit (LSB) is the coefficient a_0.

Polynomial addition of two elements $a(x)$ and $b(x)$ in $GF(2^N)$ is a simple exclusive-or (xor) of the coefficients:

$$c(x) = a(x) + b(x) = \sum_{i=0}^{N-1} (a_i \oplus b_i) x^i. \tag{2}$$

Notice that, because exclusive-or is its own inverse operation, subtraction is carried out in a same manner than addition i.e. by xorring.

Multiplication of two elements in $GF(2^N)$ (denoted by $\bullet$) is a more difficult operation than addition. First the multiplication of the elements is performed in the same way as for regular polynomials. The result of this algebraic multiplication (denoted by $\cdot$) can have a degree of $2(N-1)$ and therefore it does not fit into an element of $GF(2^N)$. Thus the result of the multiplication is the result of the algebraic multiplication modulo an irreducible polynomial $m(x)$.

For example, in the case of the Galois field $GF(2^8) : GF(2)[x]\,/\,x^8 + x^4 + x^3 + x + 1$ used in Advanced Encryption Standard (AES) [9] the multiplication of elements $0x32$ and $0x88$ is calculated as follows:

$$\begin{gathered} 0x32 = x^5 + x^4 + x \\ 0x88 = x^7 + x^3 \\ (x^5 + x^4 + x) \cdot (x^7 + x^3) = x^{12} + x^{11} + x^7 + x^4 \end{gathered}$$

Next the reduction modulo $m(x) = x^8 + x^4 + x^3 + x + 1$ is performed

$$\begin{array}{llllllllllll}
 & & & & & & & & x^4 & +x^3 & & +1 \\ \cline{3-12}
x^8+x^4+x^3+x+1 & : & x^{12} & +x^{11} & & +x^7 & & & +x^4 & & & \\
 & \oplus & x^{12} & & +x^8 & +x^7 & & +x^5 & +x^4 & & & \\ \cline{3-12}
 & & & x^{11} & +x^8 & & & +x^5 & & & & \\
 & \oplus & & x^{11} & & +x^7 & +x^6 & & +x^4 & +x^3 & & \\ \cline{3-12}
 & & & & x^8 & +x^7 & +x^6 & +x^5 & +x^4 & +x^3 & & \\
 & \oplus & & & x^8 & & & & +x^4 & +x^3 & +x & +1 \\ \cline{3-12}
\text{Result:} & & & & & x^7 & +x^6 & +x^5 & & & +x & +1
\end{array}$$

and finally the result of the multiplication is $0x32 \bullet 0x88 = 0xE3$.

Element $b(x)$ is a multiplicative inverse of an element $a(x)$ in Galois field $GF(2^N)$ if and only if

$$a(x) \bullet b(x) = \mathbf{1}. \tag{3}$$

Multiplicative inverse can be calculated for example with Euclid's greatest common divisor algorithm. Inversion is usually considered a very time consuming operation and therefore it should be avoided whenever possible. Actually, the need for several field inversions was for a long time the main reason that prevented ECC to really break into common use [27].

2.2 Normal Basis Mathematics

There are certain elliptic curve cryptography implementations which use optimal normal basis (ONB) instead of polynomial basis discussed in Section 2.2. Polynomial basis mathematics are commonly considered faster and easier to implement than ONB mathematics and they are used more commonly in elliptic curve cryptography. However, also optimal normal basis is used in several implementations and therefore it is important to understand the basics of ONB.

An element β in Galois field $GF(2^N)$ has a polynomial representation as follows (see previous section):

$$\beta = a_{N-1}x^{N-1} + \ldots + a_1 x + a_0. \tag{4}$$

A normal basis for $GF(2^N)$ can then be formed using the set [27]:

$$\{\beta^{2^{N-1}}, \ldots, \beta^{2^2}, \beta^2, \beta\}. \tag{5}$$

It can be proven that a normal basis exists for any finite field [27]. An element e in a normal basis is written as:

$$e = \sum_{i=0}^{N-1} e_i \beta^{2^i}. \tag{6}$$

Because coefficients e_i are elements in $GF(2)$ also e can be represented with N bits.

Addition of two elements is performed with a simple xor-operation of the coefficients also in a normal basis. In a normal basis squaring is just a simple rotation. There are two reasons for this:

$$1. \quad \left(\beta^{2^i}\right)^2 = \beta^{2^{i+1}} \tag{7}$$

$$2. \quad \beta^{2^N} = \beta. \tag{8}$$

This means that the squaring is very easily implemented in hardware. Squaring is performed simply by cyclically rotating the bit vector one position to the left. It should also be noticed that rotation to the right gives the square root of an element.

Multiplication over a normal basis is a bit more difficult operation. Take two elements A and B in $GF(2^N)$:

$$A = \sum_{i=0}^{N-1} a_i \beta^{2^i} \quad \text{and} \quad B = \sum_{j=0}^{N-1} b_j \beta^{2^j}. \tag{9}$$

Then multiplication $C = A \bullet B$ is given as:

$$C = A \bullet B = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} a_i b_j \beta^{2^i} \beta^{2^j}. \tag{10}$$

On the other hand also C is an element in a normal basis and therefore

$$C = \sum_{k=0}^{N-1} c_k \beta^{2^k}. \tag{11}$$

Because both equations represent the same element, the double sum in Equation (10) must match to the single sum in Equation (11). Every cross-product term maps to a sum over the basis terms [27]:

$$\beta^{2^i} \beta^{2^j} = \sum_{k=0}^{N-1} \lambda_{ijk} \beta^{2^k}. \tag{12}$$

The coefficients λ_{ijk} form a matrix which is usually called *the lambda matrix* or *the multplication table*. It can be shown that the coefficients c_k can be derived as follows:

$$c_k = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} a_i b_j \lambda_{ijk}. \tag{13}$$

It can be proven [27] that only coefficients λ_{ij0} from the lambda matrix are needed to derive c_k:

$$c_k = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} a_{i+k} b_{j+k} \lambda_{ij0}. \tag{14}$$

Equation (14) makes it possible to calculate the coefficients of C in parallel by shifting the inputs.

The number of ones in the lambda matrix indicates the measure of the hardware complexity. The number of ones is bounded between $2N-1$ and N^2 [3]. A normal basis is called optimal normal basis (ONB), if the lower bound is attained.

3 Elliptic Curves

The theory of elliptic curves is an old and deep branch of mathematics, which was for a long time considered purely academical without any practical applications. In addition to cryptography, elliptic curves are nowadays used also for factoring and for solving Fermat's Theorem [27].

In elliptic curve cryptography (ECC), an algebra is created over an elliptic curve and it is then used for cryptographic purposes. Using this algebra, two points on the elliptic curve can be added and a third point is produced as the result. Importantly for cryptography, it is very difficult to tell which two points were added [27].

In the following sections elliptic curves are discussed in detail. First elliptic curves over real numbers are introduced in Section 3.1, because the main features of elliptic curves are easier to explain and understand when more familiar real numbers are used. In Section 3.2 elliptic curves over Galois fields are discussed. These curves are actually the ones used in elliptic curve cryptography. Point multiplication over an elliptic curve is presented in Section 3.3 and it is specially important because all ECC algorithms are using it. In Section 3.4 a glance to the elliptic curve discrete logarithm problem is taken.

3.1 Elliptic Curves over Real Numbers

First elliptic curves over real numbers are considered, because it is easier to get an insight of addition and multiplication over an elliptic curve when they are explained with more familiar real number curves.

An interesting feature of elliptic curve theory is that an algebra can be created over an elliptic curve. Having two points on an elliptic curve and adding them together, a third point, which is also on the curve, is produced as the result. Importantly for cryptography, it is very difficult to say which two points were added. In fact the difficulty of this problem grows exponentially with the key length [27] (see Section 3.4).

The Weierstrass equation[2] is defined as

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{15}$$

where x and y are variables covering a plane [3]. In future this is simply called *an elliptic curve*. Notice that x and y can be complex, real, integers or any kind of field elements [27]. An elliptic curve $E_R : y^2 = x^3 + 5x^2 + 4$ over reals is presented in Figure 1. This curve will be later used for demonstrating elliptic curve algebra.

To create an algebra over an elliptic curve, addition must be defined and an *identity element* must be found. The identity element O_∞ is the point that added to any point on a curve produces the same point as the result:

$$P + O_\infty = P. \tag{16}$$

When elliptic curves are discussed the identity element is usually called *the point at infinity*. That is because if elliptic curves over real numbers are considered, this point can be thought of as lying infinitely far up the y-axis [3]. Equation (16) can also be presented in the following form:

$$P + (-P) = O_\infty. \tag{17}$$

[2]Actually this is the affine version of the Weierstass equation, but here it is simply called the Weierstrass equation. For details see Reference [3].

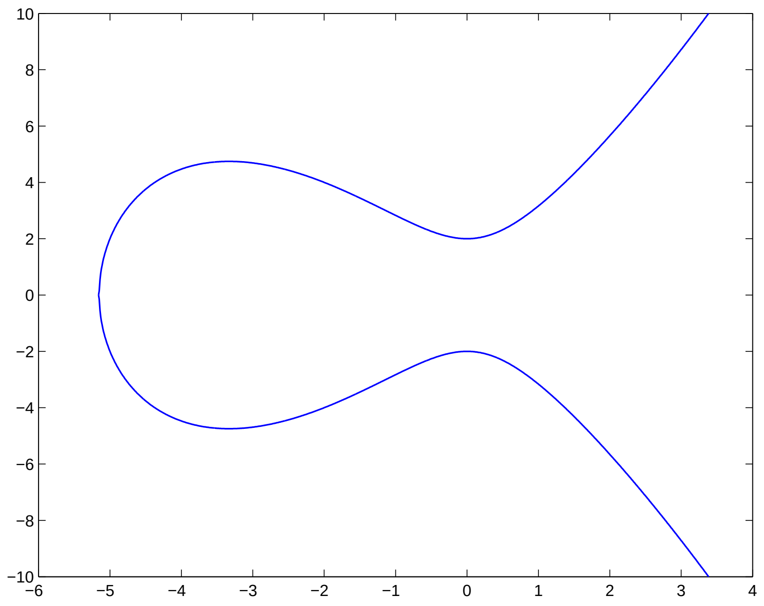

Figure 1: Elliptic curve $E_R : y^2 = x^3 + 5x^2 + 4$ over real numbers.

Having an arbitrary point $P = (x,y)$ on a curve, the point $-P$ is found for real valued E of Equation (15) with the formula:

$$-P = (x, -y). \tag{18}$$

In Figure 2 this is shown for a point P over the example curve E_R.

Addition can be defined by drawing a line between two points P and Q. The third point, which the line intersects, is here called $-R$ and now addition can be defined as follows:

$$R = P + Q. \tag{19}$$

An example of addition on elliptic curve E_R is shown in Figure 3. When Figure 2 and Equation (17) are compared, it should be clear why the identity element O_∞ is called the point at infinity.

Exact formulae for addition over real number curves defined by Equation (15) exist and they are presented in Reference [3]. Because elliptic curves over real numbers are not used for elliptic curve cryptography, these formulae will not be considered here. Equivalent formulae for elliptic curves over Galois fields are presented in Section 3.2.

A special case, where $P = Q$ in Equation (19), is discussed next. When a point is added to itself, the line used is the tangent of the curve on that point. Adding a point to itself is very important in elliptic curve cryptography and it is usually called *doubling*. As shown later in Section 3.3, many efficient point multiplication algorithms use doubling. An example of doubling the point P on the elliptic curve E_R is shown in Figure 4.

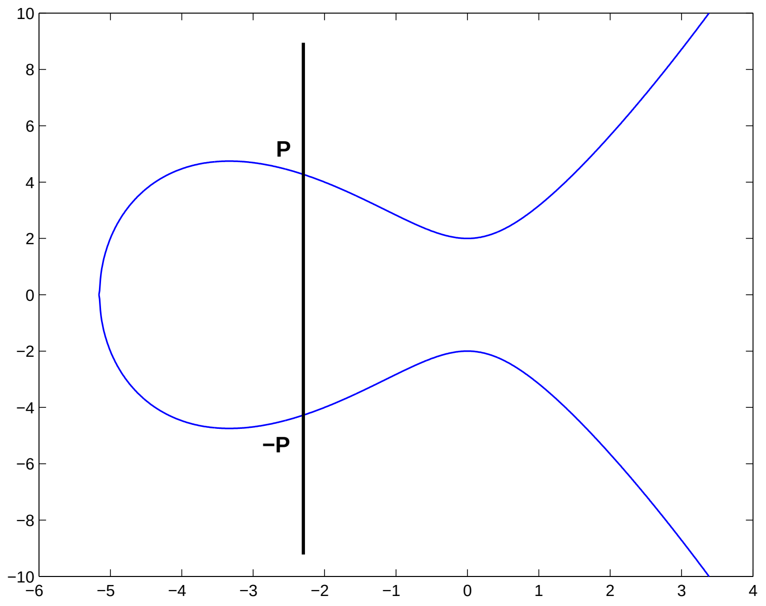

Figure 2: Points P and $-P$ on E_R.

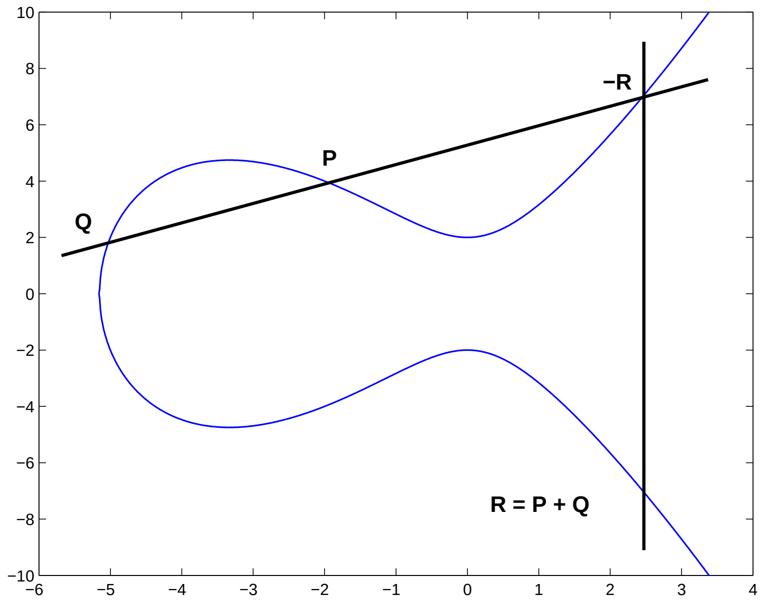

Figure 3: Addition over the example elliptic curve E_R.

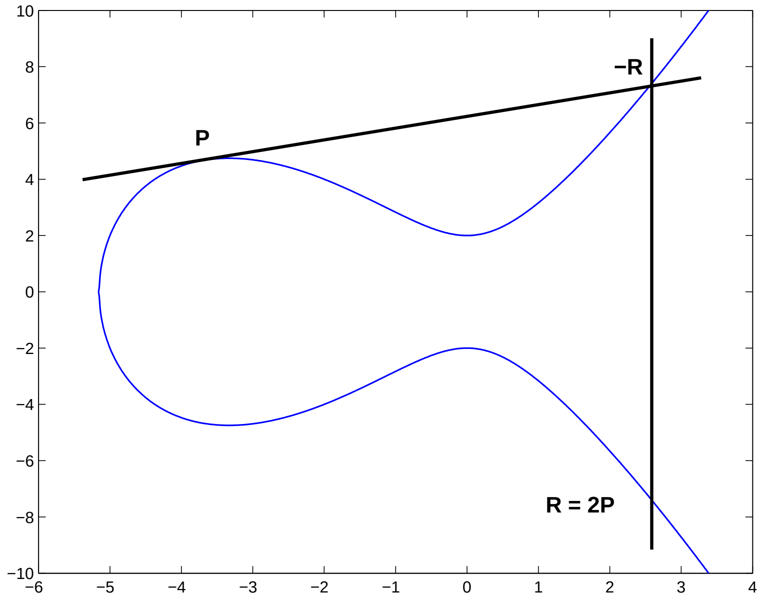

Figure 4: Adding the point P to itself on E_R.

3.2 Elliptic Curves over Galois Fields

Next elliptic curves over Galois fields are discussed. They are specially important because elliptic curve cryptosystems are defined using elliptic curves over Galois fields. Points on an elliptic curve defined by Equation (15), where the coefficients a_i are elements in $GF(2^N)$, form a finite group of order $n = \#E(GF(2^N))$ with the point addition as a group operation [3].

Curves usually used for elliptic curve cryptography are either *supersingular* or *nonsupersingular* curves. In the supersingular form $a_1 = 0$ in Equation (15). Nowadays nonsupersingular curves are preferred to supersingular curves though calculations are faster over supersingular curves [27]. The reason for this is that supersingular curves have special properties that make them less secure [3]. However, supersingular curves of form

$$y^2 + y = x^3 + a_4x + a_6 \tag{20}$$

were used in some older implementations.

As mentioned, it is recommended to use nonsupersingular curves in elliptic curve cryptography as there is no known method to attack them in less than exponential time [27]. Nonsupersingular curves are defined as

$$y^2 + xy = x^3 + a_2x^2 + a_6, \tag{21}$$

where x and y are variables in $GF(2^N)$, a_2 and a_6 are constants in $GF(2^N)$ and $a_6 \neq 0$.

The rules for addition and doubling are similar to the ones described in Section 3.1. Thus an addition of two points can be derived by drawing a line between them and the negative of the third point, which the line intersects, is the sum of the two points. However, it is very difficult (or even impossible) to draw a picture to show this graphically. The rule to negate a point $P = (x,y)$ is given as

$$-P = (x, x+y). \tag{22}$$

Addition of two points $P = (x_1,y_1)$ and $Q = (x_2,y_2)$ produces the result $R = P + Q = (x_3,y_3)$. Setting

$$\lambda = \frac{y_2+y_1}{x_2+x_1}, \quad \text{when} \quad P \neq Q \quad \text{and} \tag{23}$$

$$\lambda = \frac{x_1^2+y_1}{x_1}, \quad \text{when} \quad P = Q \tag{24}$$

x_3 and y_3 are given as follows

$$x_3 = \lambda^2 + \lambda + a_2 + x_1 + x_2, \tag{25}$$

$$y_3 = (x_1 + x_3)\lambda + x_3 + y_1. \tag{26}$$

Equations (25) and (26) can be expressed as

$$x_3 = \lambda^2 + \lambda + a_2, \tag{27}$$

$$y_3 = (\lambda + 1)x_3 + x_1^2. \tag{28}$$

when $P = Q$ because $a + a = 0$ in $GF(2^N)$. More detailed derivations of these formulae are given in Reference [3]. If $P = Q$, x_3 can also be calculated with the equation:

$$x_3 = x_1^2 + \frac{a_6}{x_1^2}. \tag{29}$$

It should be noticed that no information of the y-coordinate of P is needed in calculating the x-coordinate of $2P$. This observation can be used in the derivation of an improved method of point multiplication over elliptic curves as Julio López and Ricardo Dahab showed in [19]. This method will be presented in Section 3.3.

It is required to calculate the inverse of an element in $GF(2^N)$ in order to calculate λ as can be seen from Equations (23) and (24). This was for a long time the main reason that prevented elliptic curve cryptography to really break into the mainstream of cryptography. Many fast ways to calculate inverses have been developed since the mid-90s [27]. Some of these methods can be found for example in References [27] and [31]. Also methods, that decrease the number of field inversions, have been published. These methods usually use *projective coordinates* to represent a point on an elliptic curve.

If a point is presented in projective coordinates, three coordinates X, Y and Z are used instead of x and y used for affine coordinates. Mapping from affine coordinates to projective coordinates is done trivially by setting $X = x$, $Y = y$ and $Z = 1$.

Point multiplication (see Section 3.3) can be implemented much more efficiently in projective coordinates because no finite field inversions are needed. Mapping back to affine coordinates requires one finite field inversion. Reference [13] has a detailed explanation of different point representations and elliptic curve cryptography in general.

Point addition, point subtraction (adding of negated point) and point doubling are *point operations*. Point operations are calculated using *field operations* which are addition (or subtraction), multiplication, squaring and inversion (division) in Galois field. For example, calculating the point operation $R = P + Q$ where $P \neq Q$ using Equations (25) and (26) requires 9 field additions, 1 squaring, 2 multiplications and 1 inversion, which is a total 13 field operations. It should be noticed that computational intensivities of different field operations are not the same. Field inversion is the most time consuming operation while addition is fast and easy to perform.

3.3 Multiplication over an Elliptic Curve

A basic operation of an elliptic curve cryptosystem is *point multiplication* which is defined by formula:

$$Q = kP = \underbrace{P + P + \ldots + P}_{\text{k times}}, \tag{30}$$

where P is a point on a curve and k is an integer in the range $1 \leq k < \text{order}(P)$ [3]. In other words, multiplication over an elliptic curve means that a point P on the curve is added to itself k times. N_o is *the order of the point P*, if and only if P multiplied with N_o results in the point at infinity. This is formally described as follows:

$$\text{order}(P) = N_o \quad \Leftrightarrow \quad N_o P = O_\infty. \tag{31}$$

The integer k can be very large and therefore it is far too slow to calculate point multiplications by just adding the point P to itself k times. Thus, faster methods are needed. Several methods are considered in Reference [3] and certain methods are presented here as examples.

The simplest and oldest efficient point multiplication method is the so called *binary method* (also called the *double and add algorithm*) which relies on the binary expansion of k. In the binary form k is presented with m bits as follows:

$$k = \sum_{j=0}^{m-1} k_j 2^j. \tag{32}$$

The binary method is presented in the following pseudo code [3]:

```
INPUT: Point $P$ and integer $k = \sum_{j=0}^{m-1} k_j 2^j$
OUTPUT: $Q = kP$
$Q = O_\infty$
for (i = m-1, i >= 0, i--){
  $Q = 2Q$
  if $k_i = 1$
    $Q = Q + P$
}
return $Q$
```

In Reference [27] Michael Rosing presents a method first introduced by Neal Koblitz in [17]. This so called *Koblitz's trick* requires a calculation of *a balanced expansion* of the integer k. The balanced expansion is first derived and then doublings, summations and subtractions are performed using it.

To show how the balanced expansion is derived, an integer $k = 973$ is considered. In the binary form $k = 0b1111001101$. In the balanced expansion all strings of set bits are converted to strings of zero bits followed by -1. The balanced expansion for k is derived as follows (strings of set bits are marked with a bold font):

Binary form of k:		1	1	1	1	0	0	**1**	**1**	0	1
Intermediate result:		**1**	**1**	**1**	**1**	0	1	0	-1	0	1
Balanced expansion of k:	1	0	0	0	-1	0	1	0	-1	0	1

In the balanced expansion going to the next "bit" always requires doubling. If the "bit" is set, P is added and if the "bit" is -1, P is subtracted. If the "bit" is zero only doubling is performed. Thus multiplication $Q = kP = 973P$ can be calculated as

$$Q = (P*2*2*2*2-P)*2*2+P)*2*2-P)*2*2+P. \qquad (33)$$

As can be seen, this reduces the amount of required calculations. In addition, doublings are slightly simplier operations than summings specially if ONB mathematics are used. This can be noticed by comparing Equations (23) and (24). Subtractions are calculated by first negating the point P with Equation (22) (i.e. by xorring) and then adding the negated point $-P$.

Because a "bit" in balanced expansion may have values 0, 1 and -1, two bits are required to present one "bit" of the balanced expansion. Another drawback of this method is the extra time required to calculate the balanced expansion.

Julio López and Ricardo Dahab described an improved algorithm for the point multiplication operation in [19]. This method is here called *Montgomery point multiplication method*. It relies on the fact that, for point multiplication $Q = kP$, only the x-coordinate of P is needed if the x-coordinate of Q is calculated, as can be seen from Equation (29). The y-coordinate of Q is calculated using the x-coordinate at the end of the algorithm. The Montgomery point multiplication method is presented in the following pseudo code:

```
INPUT: Point $P = (x,y)$ and integer $k = \sum_{j=0}^{m-1} k_j 2^j$
OUTPUT: $Q = kP$
if $k = 0$ or $x = 0$ return $Q = (0,0)$
$x_1 = x$
$x_2 = x^2 + a_6/x^2$
for (i = m-2, i >= 0, i--)
  $t = x_1/(x_1 + x_2)$
  if $k_i = 1$
    $x_1 = x + t^2 + t$
    $x_2 = x_2^2 + a_6/x_2^2$
  else
    $x_1 = x_1^2 + a_6/x_1^2$
    $x_2 = x + t^2 + t$
$r_1 = x_1 + x$
$r_2 = x_2 + x$
$y_1 = r_1(r_1 r_2 + x^2 + y)/x + y$
return $Q = (x_1, y_1)$
```

Details of the derivation of the above algorithm can be found from Reference [19]. At first sight this method seems more complex than the binary method or Koblitz's trick, but noticing that all the calculations in the algorithm are field operations instead of point operations, it can be figured out why this method is faster. Also an advantage over Koblitz's trick is that no precomputations are required.

The number of required field inversion in the above algorithm can be considered a serious drawback, but the need for inversions can be significantly decreased if projective coordinates are used instead traditional affine coordinates. This method will not be discussed here in detail, but many implementations presented in Section 5 will use it and it is there called *projective version of Montgomery multiplication*.

3.4 Elliptic Curve Discrete Logarithm Problem

The basis of elliptic curve cryptography is *the elliptic curve discete logarithm problem* (ECDLP), which can be stated in the following way. Let E be an elliptic curve defined by Equation (15) and let P and Q be two points on E so that

$$Q = kP, \tag{34}$$

where k is a (large) integer. This multiplication is defined as in Section 3.3. The problem of finding k, when P and Q are given, is called the elliptic curve discrete logarithm problem.

Very detailed examinations of different methods of solving ECDLP (the MOV attack, the anomalous attack, baby step/giant step etc.) are discussed in [3], but they are not considered here in detail. It suffices to say that ECDLP is considered to require an exponential time to solve if several kinds of curves are avoided in the selection of the elliptic curve. Specially curves of *anomalous*[3] or supersingular form should be avoided. All points on an elliptic curve over $GF(q)$ form a group of order n. This group should have a subgroup of a large prime order r. This is often understood to mean a prime of more than 160 bits, which provides the security of conventional public-key cryptography using about 1000 bits [3].

4 Elliptic Curve Cryptography

The use of elliptic curves in cryptography was first proposed by Neil Koblitz [16] and Victor Miller [20] in 1985. Koblitz and Miller did not invent a new cryptographic algorithm but they implemented certain existing algorithms using elliptic curve arithmetic. Since its founding elliptic curve cryptography has been studied a lot in the academic world. The use of ellipic curves in cryptography is very inviting because shorter key lengths can be used than in the case of conventional cryptography e.g. RSA.

As already mentioned in Section 3, points on an elliptic curve over $GF(2^N)$ form a finite group of order $n = \#E(GF(2^N))$, with the point addition as a group operation. Multiplication over an elliptic curve is defined as in Section 3.3, i.e. it is performed by sequentially adding a point to itself. Multiplication is the basic operation of any elliptic curve cryptosystem and many efficient algorithms to compute it have been developed.

All elliptic curve cryptography (ECC) algorihms rely on the fact that calculating the point multiplication kP, where k is an integer and P is a point on an elliptic curve, is relatively easy and fast, but it is a very hard task to calculate k, if P and kP are given. The problem that must be solved, to calculate k, is called elliptic curve discrete logarithm problem and it requires an exponential time to solve (see Section 3.4).

There are certain known methods to attack ECC in less than exponential time if certain types of elliptic curves are used. If these types of elliptic curves are avoided, ECC is considered to require an exponential time to crack. Specially elliptic curves of anomalous and supersingular forms should be avoided [3]. It is recommended to use elliptic curves of nonsupersingular form, as presented in Equation (21), for ECC because there is no known way to attack them in sub-exponential time [3].

Elliptic curve cryptography has better security with a shorter key length than any other published public-key cryptography method. Elliptic curve cryptosystem with a 173-bit key is considered as secure as RSA using a 1024-bit key and ECC with a 313-bit key is considered as secure as 4096-bit RSA [3]. Elliptic curve cryptography is thus a very attractive alternative, specially in communication systems with limited bandwidth [2].

[3]Anomalous elliptic curves are curves for which the trace of Frobenius $t = 1$ and the order n of the group formed by the points on the elliptic curve is a prime. For details see [3].

The figures above were calculated using an approximate formula presented by Blake et al. in [3]. Let N_{CC} be the key size of conventional cryptography e.g. RSA. Then equivalent level of security is achieved using ECC with a key of N_{ECC} bits. This key length can be calculated with the formula [3]:

$$N_{ECC} = \beta N_{CC}^{1/3} (\log(N_{CC} \log 2))^{2/3} \tag{35}$$

where $\beta = 2c_0/(\log 2)^{2/3} \approx 4.91$ and $c_0 = (64/9)^{1/3} \approx 1.92$. This relation has been calculated by assuming that solving elliptic curve cryptography is of exponential complexity in relation to the field size while there are algorithms for solving conventional cryptography with sub-exponential complexity. This assumption is valid at least at the present time, because there are no known methods to attack ECC, if the curves used are selected carefully, but there are certain algorithms available to attack conventional cryptography. The relation in Equation (35) is plotted in Figure 5.

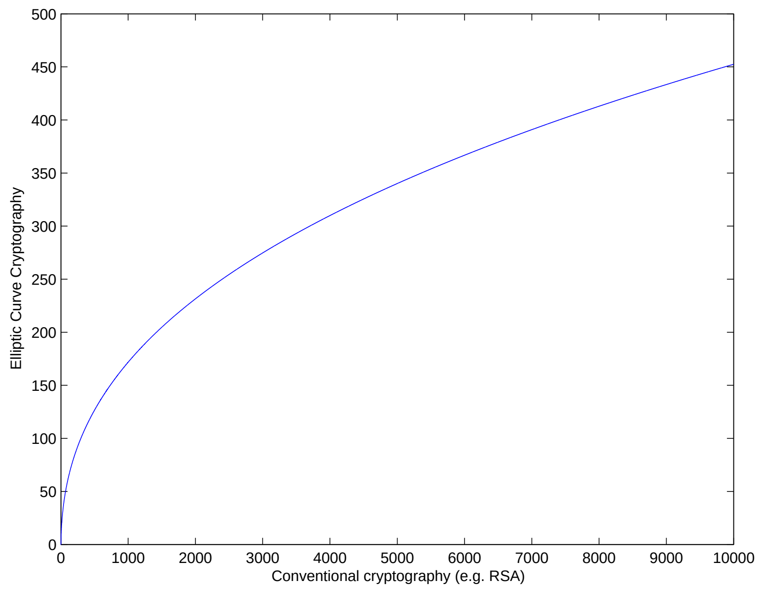

Figure 5: ECC and conventional cryptography key sizes needed for equivalent level of security [3].

As already mentioned, the point multiplication operation $Q = kP$ over an elliptic curve E is the most important operation in all elliptic curve cryptosystems. This is usually implemented so that the point P and the curve E are public information while the integer k is private. The integer k is usually called as the *private key* or the *secret key*. The result Q of the multiplication is typically also public and it is usually called the *public key*.

Next certain cryptographic algorithms using ECC will be presented. Elliptic Curve Diffie-Hellman, presented in Section 4.1, is an elliptic curve analogue of the traditional Diffie-Hellman key-exchange protocol, whereas ECDSA is the analogue of the Digital Signature Algorithm (DSA). ECDSA is considered in Section 4.2. These algorithms are presented by supposing that two persons (or computers etc.) Alice (A) and Bob (B) want to send data over an insecure channel. The eavesdropper Eve (E) is trying to read, modify or forge the messages sent my either Alice or Bob. In Section 4.3 the standardization process of ECC and certain ECC standards are presented. Certain ECC patents are presented in Section 4.4.

4.1 Elliptic Curve Diffie-Hellman (ECDH)

Diffie-Hellman was the first public-key algorithm ever invented [30]. It was proposed in 1976 by Whitfield Diffie and Martin E. Hellman in [5]. Diffie-Hellman is a key-exchange protocol, which means that it is used for changing keys that are then used by symmetric cryptography algorithms. The original Diffie-Hellman algorithm uses exponentiation over finite field, but a version using elliptic curves has been presented by the IEEE and it is described for example in [27]. This elliptic curve version is here called *elliptic curve Diffie-Hellman* (ECDH).

ECDH is described here as in [27]. Two persons, Alice (A) and Bob (B), who want to communicate with each other. Both Alice and Bob have their own private keys which are considered as random bit patterns. Let Alice's private key be k_A and let Bob's private key be k_B. First Alice and Bob agree to use a specific elliptic curve, field size and type of mathematics (polynomial basis or ONB). All this information is public.

Then a random point B on the elliptic curve is chosen and it is shared by both Alice and Bob. Thus this point is also public. After that Alice calculates the point multiplication:

$$P_A = k_A B \tag{36}$$

over the chosen elliptic curve and sends P_A to Bob. Bob calculates:

$$P_B = k_B B \tag{37}$$

and sends it to Alice. Now both Alice and Bob (and also the possible eavesdropper Eve) have both P_A and P_B. The shared secret P_S can now be calculated by both Alice and Bob. Alice calculates:

$$P_S = k_A P_B = k_A(k_B B) \tag{38}$$

and Bob calculates

$$P_S = k_B P_A = k_B(k_A B). \tag{39}$$

In order to calculate the shared secret, Eve would have to solve the elliptic curve discrete logarithm problem (see Section 3.4), because she would have to calculate either k_A from P_A or k_B from P_B.

Diffie-Hellman (and also ECDH) has a weakness because it can be attacked with the so called *man-in-the-middle attack*. Consider a situation where Eve can not only listen to the communication between Alice and Bob, but she can also modify, delete and create new messages. Now Eve can imitate Bob when talking to Alice and imitate Alice when talking to Bob [30].

Man-in-the-middle attack would work for ECDH for example in the following way: first Alice sends her public key P_A to Bob, but Eve catches it in the middle and replaces it with her own public key $P_{EA} = k_{EA}B$. When Bob sends his public key P_B to Alice, Eve replaces it with $P_{EB} = k_{EB}B$. Now there are two shared secrets P_{SA}, which is used in the communication between Alice and Eve, and P_{SB}, used by Eve and Bob. Both Alice and Bob are unaware that they are communicating through Eve. Thus Eve can decrypt all messages that are coming either from Alice or from Bob, read or modify them and then encrypt them again and send them forward to Bob or to Alice. The possibility of man-in-the-middle attack can be eliminated for example by using several communication channels [27].

4.2 Elliptic Curve Digital Signature Algorithm (ECDSA)

The *Elliptic Curve Digital Signature Algorithm* (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA) [14]. ECDSA is a standard of ISO (since 1998), ANSI (since 1999), IEEE and NIST (since 2000). ANSI X9.62 ECDSA is described here as it was presented by Johnson, Menezes and Vanstone in [14].

Digital signature schemes can be considered as the digital counterparts for handwritten signatures. It is commonly required, that it can be verified that a message is truly written by the one who claims it. A *digital signature* is a number which has been created using the signer's secret key and the contents of the message that is being signed. There are many requirements for digital signature. First, the signature must be verifiable for a third-party without the knowledge of the signer's secret key. Secondly, the signature must be linked with the message so that it cannot be copied to another message by a forger. Digital signature schemes must also be used for the verification of data integrity, i.e. the message has not been altered by unauthorized means [14].

ECDSA is an *asymmetric* digital signature scheme with an *appendix* [14]. Asymmetric means that a key pair, consisting of a secret key and a related public key, is used. Thus, the authentication can be done without the signer's secret key. Appendix means that a cryptographic hash function (SHA-1 in ECDSA, see [10] for details) is used to create a *message digest* of the message. Then this message digest is used for signature transformation instead of the message itself [14].

First, *elliptic curve domain parameters D* are chosen and calculated. The EC domain paramenters D are: field $GF(2^n)$, field basis, elliptic curve (parameters a_2 and a_6), a random base point G on the elliptic curve, the order N_0 of the point G and a cofactor $h = \#E(GF(2^n))/N_0$, where $\#E(GF(2^n))$ is the order of the group formed by points on the elliptic curve. All this information is public.

Now, the signer Alice performs *the ECDSA key pair generation process*. Public and secret keys are derived in this process. The process is performed as follows:

1. Select a random or pseudorandom integer d in the interval $1 \leq d < N_0$.
2. Compute $Q = dG$ over the selected elliptic curve.
3. The signer's public key is now the point Q and the secret key is the integer d.

To sign a message m, Alice will perform *the ECDSA signature generation process*. With the EC domain parameters D and the key pair (d, Q) the signature process for the message m is performed as follows:

1. Select a random or pseudorandom integer k in the interval $1 \leq k \leq N_0 - 1$.
2. Compute $kG = (x_1, y_1)$ and convert x_1 to an integer $\overline{x}_1$.
3. Compute $r = \overline{x}_1 \bmod N_0$. If $r = 0$, go to step 1.
4. Compute $k^{-1} \bmod N_0$.
5. Perform the hash function for message m: SHA-1(m), and convert the result to an integer e.
6. Compute $s = k^{-1}(e + dr) \bmod N_0$. If $s = 0$, go to step 1.
7. The signer's signature for the message m is (r, s).

To verify Alice's signature of the message m, Bob will need the EC domain parameters D and the public key Q which are all public information. Then *the ECDSA signature verification process* is performed as follows:

1. Verify that r and s are integers in the interval $1 \leq r, s < N_0$.
2. Perform the hash function for message m: SHA-1(m) and convert the result to an integer e.
3. Compute $w = s^{-1} \bmod N_0$.
4. Compute $u_1 = ew \bmod N_0$, and $u_2 = rw \bmod N_0$.
5. Compute $X = u_1G + u_2Q = (x_1, y_1)$.
6. If $X = O_\infty$, the signature must be rejected. Else, convert x_1 to an integer $\overline{x}_1$, and compute $v = \overline{x}_1 \bmod N_0$.
7. Accept the signature if and only if $v = r$.

Next, it is proven that the above process really verifies the authentication of the signature. If a signature (r, s) on a message m was truly made by the correct signer, then $s = k^{-1}(e + dr) \bmod N_0$. Multiplicating both sides with ks^{-1} yields:

$$k = s^{-1}(e + dr) = s^{-1}e + s^{-1}dr = we + wdr = u_1 + u_2d \pmod{N_0}. \tag{40}$$

By combining the public key $Q = dG$ and $X = u_1G + u_2Q$, a result $X = u_1G + u_2dG$ is attained. Thus, from the above equation it can be seen that $X = kG$ and therefore $v = r$ is required for the verification [14].

If Eve wants to forge Alice's signature, she must have Alice's secret key d so that she can herself perform the ECDSA signature generation process. Again, in order to calculate d from the public key $Q = dG$, Eve will have to solve the elliptic curve discrete logarithm problem (ECDLP). Because the message digest of the message m is used for the derivation of the signature, Eve cannot use this signature to other messages and claim that they were signed by Alice. Also notice that if Eve could somehow alter the contents of the message m, the verification would fail, because the message digest e derived during the verification process would be different to the message digest e used for the signature generation process.

4.3 Standards

The use of elliptic curves for cryptography was introduced first back in 1985. However, the first standards for ECC were introduced as late as in the end of the 90s. During this quite long period, many kinds of elliptic curve cryptosystems using different types of elliptic curves were suggested and tested. This can be considered as a major benefit, because now it is pretty certain, that the curves used for elliptic curve cryptography today are secure. For example, supersingular curves are now known to be insecure, though they first seemed very inviting because calculations are faster over them.

Elliptic curve cryptography is a part of the IEEE 1363-2000 standard which is a standard specification for public-key cryptography introduced in 2000. In addition to elliptic curve cryptography, IEEE 1363 also defines public-key cryptography schemes using traditional discrete logarithm in the group of remainders modulo a prime (DL) and integer factorization (IF). The IEEE 1363 standard includes the following algorithms for both DL and ECC: Diffie-Hellman key agreement (see Section 4.1) allowing up to two key pairs from each party, Menezes-Qu-Vanstone key agreement, which requires two key pairs from each party, DSA Signatures (see Section 4.2), with SHA-1 and RIPEMD-160 as hash functions, Nyberg-Rueppel Signatures with appendix, with SHA-1 and RIPEMD-160 as hash functions. These algorithms are analogous for both DL and ECC. The only main difference is the underlying group. For IF family, IEEE 1363 includes following algorithms: RSA encryption with Optimal Asymmetric Encryption Padding (OAEP), RSA signature with appendix, and Rabin-Williams equivalents of the above RSA signatures.

As already declared in Section 4.2, where the Elliptic Curve Digital Signature Algorithm (ECDSA) was considered in detail, ECDSA is the elliptic curve analogue of the Digital Signature Algorithm (DSA) [14]. In addition to IEEE, ECDSA is also a standard of ISO, ANSI and NIST. Also Elliptic Curve Diffie-Hellman (ECDH) presented in Section 4.1 is included in many standards. The Menezes-Qu-Vanstone (MQV) key agreement scheme is more advanced than ECDH and it prevents the possibility of man-in-the-middle attack [27]. Nyberg-Rueppel is a signature scheme and it is fairly similar to ECDSA. MQV and Nyberg-Rueppel are not presented here and interested readers should consult [27] for details.

NIST has published a set of elliptic curves for cryptography use in ”Recommended Elliptic Curves for Federal Government Use” (available e.g. in Appendix 6 in [8]). There are curves over both prime fields $GF(p)$ and over binary fields $GF(2^m)$. For binary fields normal basis and polynomial basis are both supported. Two kinds of curves are given: pseudo-random curves, whose coefficients are generated from the output of a seeded cryptographic hash function (SHA-1), and Koblitz curves (anomalous curves) whose coefficients and the underlying field are selected to optimize elliptic curve operations. The pseudo-random curves over binary fields have the form:

$$y^2 + xy = x^3 + x^2 + b, \tag{41}$$

and Koblitz curves have the form:

$$y^2 + xy = x^3 + ax^2 + 1. \tag{42}$$

For example a pseudo-random curve called Curve B-163 over $GF(2^{163})$, for which the irreducible polynomial is a pentanomial $m(x) = x^{163} + x^7 + x^6 + x^3 + 1$, is given as follows:

- the coefficient b in Equation (41) is
  $b = 0x\ 2\ 0a601907\ b8c953ca\ 1481eb10\ 512f7874\ 4a3205fd$
- the x-coordinate of the base point G in the point multiplication kG is
  $G_x = 0x\ 3\ f0eba162\ 86a2d57e\ a0991168\ d4994637\ e8343e36$
- the y-coordinate of G is
  $G_y = 0x\ 0\ d51fbc6c\ 71a0094f\ a2cdd545\ b11c5c0c\ 797324f1$
- the order N_o of the point G is
  $N_o = 5846006549323611672814742442876390689256843201587$

It should be noticed, that Koblitz curves should not be used in applications requiring very high security, because Blake et al. recommended in [3] that anomalous curves should be avoided as they are vulnarable to anomalous attack.

Standards for Efficient Cryptography Group (SECG) has published ECC standards SEC 1: Elliptic Curve Cryptography [25] and SEC 2: Recommended Elliptic Curve Domain Parameters [26] in 2000. SEC 1 includes ECDSA as a signature scheme, Elliptic Curve Integrated Encryption Scheme (ECIES) as an encryption scheme and ECDH and MQV as key agreement schemes. SEC 2 defines a set of elliptic curves over both prime and binary fields. The curves over binary fields include both random and Koblitz curves. These curves are defined only over polynomial basis.

There are still some standards concerning ECC under development. RSA Security Inc. started developing a standard PKCS #13 for elliptic curve cryptography in 1998 and the standardization process of this standard is still unfinished. This standard will include several different schemes using ECC, e.g. ECDSA will also be included in this standard. Overall, it can be said that this standard will be quite similar to IEEE 1363-2000. Furher information can be found on RSA Security's web page [29].

4.4 Patents

There are various patents related to elliptic curve cryptography. Some of these define methods for efficient finite field multiplication (e.g. US patent 6,389,442: Efficient finite field multiplication in normal basis) and therefore they can be used in addition to ECC also in other applications using finite fields. Certain finite field processors have also been patented (e.g. US patent 6,349,318: Arithmetic processor for finite field and module integer arithmetic operations). Elliptic curve encryption methods have also been patented (e.g. US patent 6,480,606: Elliptic curve encryption method and system).

5 Implementations of Elliptic Curve Cryptogaphy

Field Programmable Gate Arrays (FPGAs) are almost ideal platforms for public-key algorithms based on elliptic curve cryptography for several reasons. Considerable speed increases can be achieved using FPGAs when compared to software implementations. Accelerations from 2 times up to over 35 times have been reported. On the other hand, the flexibility and cost efficiency of FPGAs make them a better alternative than ASICs, especially if the manufacturing volumes are low or medium.

FPGA-based cryptosystems avoid various problems that are faced if ASIC-based cryptosystems are used. As is commonly known, a cryptographic algorithm is secure only as long as there is no effective attack available. If an effective attack is found, the algorithm must be replaced with a more secure one. In server use, cryptographic algorithms must be often changed because different clients may use different cryptographic algorithms. Thus the reprogrammability of FPGAs is a major benefit compared to ASICs which usually cannot be altered after they have been manufactured.

The most important operation of all elliptic curve cryptosystems is the point multplication over an elliptic curve, which was discussed in detail in Section 3.3. The point multiplication operation can be divided hierachically into three levels [2]:

1. **Point multiplication methods:** Selection of the point multiplication method e.g. Binary method, Koblitz's trick, Montgomery point multiplication etc.

2. **Point addition and doubling:** Selection of the point representation method. Affine, projective, Jacobian etc.

3. **Finite field arithmetic:** Selection of basis (polynomial, normal basis etc.), multiplier and inverter structures, etc.

Register transfer level could be considered as the fourth level below the finite field arithmetic level. Optimizations are needed in every level to achieve the best result. Choices made on the lower level may affect top levels, e.g. if ONB field arithmetics are chosen, squarings are easy to implement and methods emphasizing squarings should be preferred.

There are several published FPGA-based hardware implementations. Many of these concentrate on implementing only the point multiplication operation instead of a complete ECC algorithm. Also many (co)processors for elliptic curve cryptography have been published. Certain implementations can be found from References [1], [2], [7], [12], [15], [18], [21] and [23].

5.1 Published ECC Implementations on FPGAs

Next certain FPGA-based ECC implementations are presented, to give a better insight into the different implementation issues. On the following pages there will appear certain terms and concepts that are not desribed in this report. If a more detailed information of these is needed, the reader is referred to search information from the paper at issue.

Bednara, Daldrup, von zur Gathen, Shokrollahi, Teich: Reconfigurable Implementation of Elliptic Curve Crypto Algorithms [2]

Bednara et al. present a reconfigurable ECC implementation in [2]. Their implementation is designed so that it can be easily reconfigured, which enables changing one can change for example from polynomial basis to optimal normal basis without changing the whole design. Only certain components of the design must be altered.

Several implementations were made with different choises of point multiplication method, point addition and doubling methods and finite field arithmetic. The implementations were implemented using Xilinx Virtex XCV1000-BG560-4 device. All of them use a 50 MHz clock (clock period 20 ns).

The fastest point multiplication operation, achieved by using sequential multipliers, uses two finite field multipliers, addition-subtraction chains as the point multiplication method and Montgomery point representation (projective coordinates). 110207 clock cycles are required to calculate the point multiplication and mapping back to affine coordinates (from Montgomery coordinates) requires 3248 cycles. The total time required is 2.27 ms. The time required for calculating a point multiplication is not constant as it depends on the number of ones in the multipliers. Bednara and others assumed that the number of ones is $\frac{n}{3}$.

If optimal normal basis and Massay-Omura parallel multipliers are used, the fastest implementation can perform point multiplication operation in 0.27 ms. This implementation uses one multiplier, addition-subtraction chains as the point multiplication method and Montgomery point representation. The point multiplication is calculated in only 9550 clock cycles and conversion back to affine coordinates requires 310 cycles. Unfortunately, this kind of implementation is practically unusable because the area requirements of the multiplier are enormous. Massey-Omura multiplier of this size requires as much as 24150 slices.

Gura, Eberle, Shantz:
Generic Implementations of Elliptic Curve Cryptography using Partial Reduction [12]

Gura et al. concentrate on generic implementation of elliptic curve cryptography (i.e. arbitrary elliptic curves of specific form can be used) instead of implementing only named curves as in most other published implementations. Though not as fast as implementations optimized for specific curves, generic implementation is an attractive alternative in situations where support for infrequently used curves is needed or where curves are not known beforehand.

Relatively fast generic curve implementations are made possible by novel reduction algorithms presented in the paper. These algorithms are used for irreducible polynomial reduction in the field multiplication operation (see Section 2.1). Field sizes are very large in elliptic curve cryptography and therefore traditional reduction methods are very time consuming, specially in generic curve implementations where efficient optimizations cannot be done because the field size and irreducible polynomial may change. Reduction algorithms presented in [12] are called *full reduction* and *partial reduction*, but they are far too complex to be presented here. Thus interested readers should see the paper for details.

Three kinds of implementations are presented in the paper. The first one implements named curves over specified fields, $GF(2^{163})$, $GF(2^{193})$ and $GF(2^{233})$, as recommended by SECG (Standards for Efficient Cryptography Group) in [26]. The second one implements generic curves and uses full reduction for field multiplications. The third one also implements generic curves but uses partial reduction for field multiplications. Designs were prototyped using Xilinx Virtex-E XCV2000E-FG680-7 device for which a clock frequency of 66.4 MHz was achieved.

The first implementation (named curves) is expectedly the fastest as it can be efficiently optimized. It can perform Montgomery point multiplication in 0.14 ms if the smallest field $GF(2^{163})$ is used. Generic curve implementatios are slightly slower. The implementation using partial reduction is faster of those two and it performs point multiplication on a curve over $GF(2^{163})$ in 0.93 ms. The slowest implementation is the one using full reduction. It performs point multiplication in 1.55 ms. Notice that though only values for point multiplication over specified curves are given in the paper, the generic implementations can be used for all curves over $GF(2^m)$ where $m < 256$. The named curve implementation can be used only for point multiplications over a named curve.

Orlando, Paar:
A High-Performance Reconfigurable Elliptic Curve Processor for $\mathbf{GF(2^m)}$ [23]

Orlando and Paar present in [23] a reconfigurable elliptic curve processor architecture for $GF(2^m)$. This elliptic curve processor (ECP) exploits the abilities of reconfigurable hardware to deliver optimized circuitry for different elliptic curves and Galois fields. The ECP uses bit-parallel squarers and a digit-serial multiplier and two programmable processors. The squarer and multiplier architectures can be optimized for any field order or irreducible polynomial through reconfiguration.

Three different implementations are presented. The difference between them is the use different digit sizes $D = 4$, $D = 8$ and $D = 16$ for digit-serial multiplier (see the paper for details). All of them were programmed to use either the projective version of Montgomery multiplication or the double-and-add method in projective coordinates. For digit-serial multipliers the number of required clock periods in field multiplication depends on the chosen digit size D and on the field size m. For all D and m, the projective version of Montgomery multiplication is faster than the double-and-add method.

The design was implemented on Xilinx Virtex-E XCV400E-BG432-8 device. The following parameters were used: Galois field $GF(2^{167})$ ($m = 167$), $D = 16$, Montgomery multiplication. The implementation has a clock frequency of 76.7 MHz and it requires 3002 LUTs (about 1500 slices), 1769 flip-flops and 10 Block-RAMs. It can perform point multiplication kP in 0.21 ms when the number of ones in the integer k is assumed to be close to half of the size of k.

Kerins, Popovici, Marnane, Fitzpatrick:
Fully Parametrizable Elliptic Curve Cryptography Processor
over $\mathbf{GF(2^m)}$ [15]

The implementation presented by Kerins et al. in [15] is a finite field processor which can be used for any application using Galois fields with polynomial basis. This so called ABC processor calculates

$$u = \frac{ab}{c} \bmod p \tag{43}$$

where $a, b, c \in GF(2^m)$ and p is an irreducible polynomial of degree m. This calculation requires $2m$ clock cycles. Calculations of above form are specially important in elliptic curve cryptography, because point addition and point doubling require them as can be seen from Equations (23) and (24).

The main advance of using the ABC processor is that field sizes and irreducible polynomials can be easily and fastly changed because the FPGA device does not have to be reprogrammed. This is a substantial benefit in many applications requiring repeated changes of elliptic curves and field sizes. At the time, only the field operation defined by Equation (43) is accelerated by hardware and all other operations are performed in software.

The ABC processor was implemented on Xilinx Virtex XCV2000-BG560-6 device. It requires 4048 slices and achieves a clock frequency of 40 MHz. A point multiplication accelerated with the ABC processor takes 5.1 ms on elliptic curve over $GF(2^{151})$, 6.9 ms over $GF(2^{176})$, 8.2 ms over $GF(2^{191})$ and 12.8 ms over $GF(2^{239})$.

The authors are investigating the possibility of implementing more logic of elliptic curve cryptography on FPGA which would decrease the required amount of software operations. This would naturally speed up the point multiplication operation significantly. At the present time, this implementation can be considered only as a semi-hardware implementation of elliptic curve cryptography. Thus it is not as fast as the fastest fully hardware accelerated implementations. However, the possibility of changing field sizes and irreducible polynomials on the fly, without reprogramming or reconfiguration, is inviting.

Okada, Torii, Itoh, Takenaka:
Implementation of Elliptic Curve Cryptographic Coprocessor over GF(2^m) on an FPGA [21]

Okada et al. present in [21] a coprocessor architecture for elliptic curve cryptography. It is based on a new kind of finite field multiplier architecture which enables multiplication of any bit length m by using a data conversion method. This method handles the data, that is being multiplied, so that it can be presented with $m+\alpha$ bits where α is the smallest integer for which $m+\alpha = n_1 w_1$ is satisfied (n_1 is an integer and w_1 is word size). This conversion is needed because m is a prime and the partial multiplication algorithm presented in the paper requires that the elements to be multiplied can be split into n_1 w_1-length parts. This partial multiplication is an expansion of the super-serial multiplication presented by Orlando and Paar in [22].

The design was implemented on Altera EPF10K250AGC599-2 FPGA device and it operates at 3 MHz. The implementation is capable of calculating elliptic curve point multiplication over Galois fields of an order of 163-bits or less. The point multiplication on a pseudo-random curve requires 80.3 ms and on a specific Koblitz curve 45.6 ms is required.

The ECC processor was also designed and simulated for ASICs. This implementation can calculate point scalar multiplications over up to 572-bit curves. The CE71 series of 0.25 μm ASIC was used in the simulation and it operated at 66 MHz and had a hardware size of about 165 kgates. The point multiplication on a 163-bit pseudo-random curve requires 1.1 ms and on a Koblitz curve only 0.65 ms. Different finite field sizes $m = 163, 233, 283, 409, 571$ were simulated. For 571-bit pseudo-random curve 22 ms are required to perform a point multiplication operation.

Ernst, Jung, Madlener, Huss, Blümel:
A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over GF(2^n) [7]

Ernst et al. present in [7] a finite field (FF) processor which can be used for elliptic curve cryptography. This FF processor is implemented by using a multi-segment Karutsaba multiplication which was originally presented for n-bit numbers but can also be used also for multiplication of two Galois field elements. The authors present a novel method for irreducible polynomial reduction. This method requires that the irreducible polynomial is a trinomial i.e. it has three non-zero terms:

$$m(x) = x^n + x^b + 1 \tag{44}$$

where $n > b > 0$. This requirement of course restricts the use of the method but fortunately many of the fields, commonly used for elliptic curve cryptography, have trinomials as irreducible polynomials.

The FF processor was implemented on the Atmel AT94K FPSLIC hardware platform. This product family integrates FPGA resources, an AVR 8-bit RISC microcontroller core, several peripherals and up to 36 kB SRAM within a single chip. The FPSLIC operates at 12 MHz.

Three different implementations were made. One of them uses only software resources of the FPSLIC and is therefore the slowest one. It can calculate point multiplication in 396 ms. The fastest of them implements multiplication, addition and squaring completely in hardware and performs point multiplication in 10.9 ms. All implementations use Galois field $GF(2^{113})$.

Leung, Ma, Wong, Leong:
FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor [18]

Leung et al. present an elliptic curve processor (ECP) in [18]. It uses normal basis arithmetics (see Section 2.2) instead of polynomial basis commonly used. The ECP is implemented similarly to the traditional microcoded central processing unit (CPU) and therefore it consists of an arithmetic logic unit (ALU), register file, a microcoded sequencer and microcode storage. Major differences between this ECP architecture and conventional CPU are that datapath is n bits wide and ALU performs operations in $GF(2^n)$ instead of integer arihmetic. The ECP uses binary method (see Section 3.3) as a point multiplication algorithm.

The ECP was implemented on an Annapolis Micro Systems Wildcard board which has a Xilinx Virtex XCV300-4 FPGA device on it. Three different values of n were implemented. The required number of clock cycles needed to perform point multiplication over an elliptic curve depends on n. For $n = 113$, implementation requires 1290 slices, operates at 45 MHz and performs point multiplication in 166783 clock cycles. The corresponding values for $n = 155$ and for $n = 281$ are: 1567 and 2622 slices, 36 MHz and 33 MHz, 246443 and 474504 clock cycles.

The point multiplication requires 3.7 ms for $n = 113$, 6.8 ms for $n = 155$ and 14.4 ms for $n = 281$. Comparisons to a software implementation running on a 270 MHz Sun Ultra-5 with 512 MB RAM were made and speed-ups between 27–36 times were reported. The implementation used was Michael Rosing's optimized software implementations of an optimal normal basis elliptic curve package [27].

5.2 Software implementations

There are many reported implementations of elliptic curve cryptography over finite fields. Most of these implementations focus on a single cryptographic application, e.g. ECDSA. Most hardware implementations implement only the point multiplication operation over an elliptic curve and therefore the comparison of software and hardware implementations is not straightforward. However, the point multiplication operation is the most time consuming operation in elliptic curve cryptography.

A detailed study of different implementation issues of ECC on software is presented in [13]. As an example of software implementations, an implementation by Hankerson et al. is presented. Various methods of point multiplication and field operations are introduced and analyzed in the paper. They also implemented these methods on a Pentium II 400 MHz workstation using the C-programming language. These software implementations use NIST-recommended curves (both random and Koblitz curves). These recommended curves can be found for example from Reference [8]. The fastest random-curve point multiplication, when memory was not constrained, was achieved using the so called Fixed-base comb point multiplication algorithm. Fixed-base comb point multiplication is an algorithm that requires that the point P in the point multiplication kP is fixed and then it performs certain precomputations which speed up the point multiplication. The fastest Koblitz-curve implementatation uses the so called Fixed-base window TNAF point multiplication method which is developed specially for Koblitz-curves (see the paper for details).

The random-curve implementation can perform point multiplication operation in 1.683 ms over $GF(2^{163})$, in 3.966 ms over $GF(2^{233})$ and in 5.919 ms over $GF(2^{283})$. For Koblitz-curves these numbers are 1.176 ms over $GF(2^{163})$, 2.243 ms over $GF(2^{233})$ and 3.330 ms over $GF(2^{283})$. These timings are quite impressive, but they are still over ten times slower than the fastest FPGA implementations over fields of the same size [12].

5.3 Master's Thesis Conserning ECC

Certain master's thesis have also been written of ECC. Henna Pietiläinen implemented and compared ECC and RSA signature and encryption schemes on smart cards using Java [24]. She found out that ECC is faster than RSA in signing and decryption, but it is slower than RSA in encryption and signature verification. Because of smaller key sizes, ECC requires less space to store keys and is therefore suitable for smart card solutions. Tommi Elo implemented ECDSA on Java smart card and compared the results of smart card implementation to a version on workstation in [6]. Martin Rosner implemented elliptic curve point multiplication on an FPGA in [28]. The implementation was designed to be easily reconfigurable, so that field sizes, curve coefficients etc. can be changed by reconfiguring the FPGA. Rosner used composite Galois fields $GF((2^n)^m)$ in his design which are now considered insecure because they are vulnarable to Weil decent attack [23] [32].

5.4 Summary

The comparison of different elliptic curve cryptography (ECC) implementations is very difficult, because there is so much variation between the implementations. This is mainly because there have not been any standards available until recently. Thus, all implementations implement different curves over various field sizes and field basis.

Information of implementations presented previously is gathered to Table 1. As can be seen there is a lot of variation between implementation techniques. Most of the implementations are implemented on Xilinx Virtex(-E) devices but there are also more exotic platforms for elliptic curve implementations, e.g. implementation on FPSLIC by Ernst et al. [7]. Also certain ASIC implementations have been published. Basically, it can be said that ASIC implementations are little faster than FPGA implementations but they are usually designed only for named curves over specific fields and therefore they are not as flexible as the implementations presented here.

Most of the implementations use polynomial basis (PB) as the field basis, because multiplication is easier to implement in PB than in ONB. The Massey-Omura multiplication over ONB is very fast but it requires huge amounts of FPGA resources as Bednara et al. presented in [2]. Because the field sizes are so large, many tricks are also needed to efficiently implement multiplications over PB. Many methods that split the reduction with the irreducible polynomial (see Section 2.1) into smaller parts are presented and implemented e.g. in [12] and [23].

Another important choice is the point representation. The traditional affine representation, where a point is represented using two coordinates x and y is an easy and understandable way to implement ECC. The disadvantage of this representation is that a very time consuming field inversion operation is needed for every point addition and point doubling (see Section 3.2). Therefore it is more efficient to use projective coordinates to represent a point on an elliptic curve i.e. the point is presented with three coordinates X, Y and Z instead of just two. With projective coordinates, no inversions are needed in addition and doubling but one inversion is needed when the point is mapped from projective coordinates back to affine coordinates.

The time required for the point multiplication operation also depends on the choice of the point multiplication algorithm. Many implementations use the binary method (see Section 3.3) as the point multiplication algorithm. The binary method is simple and easy to implement but Montgomery point multiplication in projective

Implementation	Device	Clock frequency (MHz)	Required Resources (e.g. slices)	Field size (m) [a]	Basis PB / ONB	Multiplier Architecture LSRF / DSM / KM / MOM / other [b]	Coordinates AC / PC [c]	Point mult. algorithm BM / MPM / ASC[d]	Point mult. time (ms)	Additional information
Bednara et al. [2]	Xilinx Virtex XCV1000-4	50	n.a.	191	PB	LFSR	PC	ASC	2.27	
	”	50	n.a.	191	PB	LFSR	PC	BM	3.72	
	” (does not fi t in the device)	50	n.a.	191	ONB	MOM	PC	ASC	0.27	Huge multiplier
Gura et al. [12]	Xilinx Virtex-E XCV2000E-7	66.4	n.a.	163	PB	hardwired	PC	MPM	0.14	Named curves
	”	66.4	n.a.	193	PB	hardwired	PC	MPM	0.19	”
	”	66.4	n.a.	233	PB	hardwired	PC	MPM	0.23	”
	”	66.4	n.a.	163 ($\leq$255)	PB	full red.	PC	MPM	1.55	Full reduction
	”	66.4	n.a.	193 ($\leq$255)	PB	full red.	PC	MPM	1.84	”
	”	66.4	n.a.	233 ($\leq$255)	PB	full red.	PC	MPM	2.22	”
	”	66.4	n.a.	163 ($\leq$255)	PB	part. red.	PC	MPM	0.93	Partial reduction
	”	66.4	n.a.	193 ($\leq$255)	PB	part. red.	PC	MPM	1.10	”
	”	66.4	n.a.	233 ($\leq$255)	PB	part. red.	PC	MPM	1.32	”
Orlando, Paar [23]	Xilinx Virtex-E XCV400E-8	85.7	ca. 815	167 ($\leq$255)	PB	DSM ($D = 4$)	PC	BM	0.96	
	”	85.7	ca. 815	167 ($\leq$255)	PB	DSM ($D = 4$)	PC	MPM	0.55	
	”	74.5	ca. 1070	167 ($\leq$255)	PB	DSM ($D = 8$)	PC	BM	0.61	
	”	74.5	ca. 1070	167 ($\leq$255)	PB	DSM ($D = 8$)	PC	MPM	0.35	
	”	76.7	ca. 1500	167 ($\leq$255)	PB	DSM ($D = 16$)	PC	BM	0.36	
	”	76.7	ca. 1500	167 ($\leq$255)	PB	DSM ($D = 16$)	PC	MPM	0.21	
Kerins et al. [15]	Xilinx Virtex XCV2000-6	40	4048	151	PB	n.a.	AC	BM	5.1	
	”	40	4048	176	PB	n.a.	AC	BM	6.9	
	”	40	4048	191	PB	n.a.	AC	BM	8.2	
	”	40	4048	239	PB	n.a.	AC	BM	12.8	
Okada et al. [21]	Altera EPF10K250AGC599-2	3	n.a.	163 ($\leq$163)	PB	partial	n.a.	n.a.	80.3	Pseudo-random curves
	”	3	n.a.	163 ($\leq$163)	PB	partial	n.a.	n.a.	45.6	Koblitz curves
	CE71 0.25μm ASIC	66	165 kgates	163 ($\leq$572)	PB	partial	n.a.	n.a.	1.1	Pseudo-random curves
	”	66	165 kgates	163 ($\leq$572)	PB	partial	n.a.	n.a.	0.65	Koblitz curves
	”	66	165 kgates	571 ($\leq$572)	PB	partial	n.a.	n.a.	22	Pseudo-random curves
Ernst et al. [7]	FPSLIC (Atmel AT40K)	12	ca. 2210 lc.	113	PB	KM	PC	BM	10.9	
Leung et al. [18]	Xilinx Virtex XCV300-5	45	1290	113	ONB	λ-matrix	AC	BM	3.7	
	”	36	1567	155	ONB	λ-matrix	AC	BM	6.8	
	”	33	2622	281	ONB	λ-matrix	AC	BM	14.4	
Hankerson et al. [13]	Pentium II 400 MHz	400	n.a.	163	PB	n.a. ?	PC	Fixed-base comb	1.68	Random curves
	”	400	n.a.	163	PB	n.a. ?	PC	Fixed-base TNAF	1.18	Koblitz curves
	”	400	n.a.	233	PB	n.a. ?	PC	Fixed-base comb	3.97	Random curves
	”	400	n.a.	233	PB	n.a. ?	PC	Fixed-base TNAF	2.24	Koblitz curves
	”	400	n.a.	283	PB	n.a. ?	PC	Fixed-base comb	5.92	Random curves
	”	400	n.a.	283	PB	n.a. ?	PC	Fixed-base TNAF	3.33	Koblitz curves

Table 1: Elliptic Curve Cryptography Implementations

[a] The size of the field $GF(2^m)$ used for point multiplication time analysis. The number in parantheses (e.g. (<256)) denotes that the same design can be used for field sizes of up to that number. (Reconfiguration may be needed.)

[b] LSFR = Linear Feedback Shift Register, DSM = Digit-Serial Multiplier, KM = Karatsuba Multiplier, MOM = Massey-Omura Multiplier

[c] AC = Affine Coordinates, PC = Some kind of Projective Coordinates (e.g. López-Dahab)

[d] BM = Minary Method, MPM = Montgomery point multiplication, ASC = Addition-Subtraction Chains

coordinates has proven to be more efficient [23]. In Montgomery point multiplication in projective coordinates the X- and Z-coordinates are calculated without information of the Y-coordinate and the Y-coordinate of the result is calculated from the X- and Z-coordinate at the end of the multiplication.

Because there are many different elliptic curves and different field sizes in use, it is very important that an implementation can be easily used for all kinds of variations of elliptic curve cryptography. The most important feature is that many different field sizes can be used, because the level of security required determines the needed field size. In other words, if less security is needed, it suffices to use a smaller field size, which makes the calculation of the point multiplication faster and more point multiplications can be performed in a second. Later, the same implementation may need to be used for an application requiring more security. Then it is important, that the field size can be changed without needing to change the whole design.

Many of the implementations presented above can be modified by using the partial reconfigurability of FPGAs i.e. only some parts of the implementation are reprogrammed while the rest remains unchanged. This is a faster way to modify a design than a complete reprogramming. If frequent changing of different field sizes is required, it is too time consuming to reconfigure the device every time, and the support for different field sizes must be included into the design, which of course requires more resources.

6 Conclusions

A survey of elliptic curve cryptogaphy (ECC) and its implementations on FPGAs has been presented. First, basic mathematics of Galois (finite) fields were discussed. Next, elliptic curves were introduced and basic arithmetic operations (addition, subtraction and multiplication) over them were defined. The use of elliptic curves for cryptogaphy and algorithms using ECC were presented in Section 4. Implementation issues of ECC and implementations on FPGAs were introduced in Section 5.

The mathematics of ECC are considerably deeper and more difficult than the mathematics used for conventional cryptography. On the other side, elliptic curves have better security with shorter key lengths than any other public-key cryptography. Similar levels of security are achieved with ECC using a 173-bit key and RSA using 1024-bit key [3]. The small key size is a major benefit in applications with limited bandwidth or limited memory resources. ECC implementations also usually require less hardware resources than RSA implementations [18]. This is mostly because of smaller key sizes.

Several choices have to be made when ECC is implemented. The field size and type of field basis must be chosen. The required level of security is the main factor which defines the needed field size. When the field size increases, also the time required for elliptic curve operations increases as well as the size of the implementation. The type of field basis depends much on the application. The polynomial basis is the most commonly used basis as it has proven to be faster than optimal normal basis (ONB). Though squarings are very fast over ONB, specially in hardware implementations, multiplications are faster and easier to implement when polynomial basis is used.

Another important choice is the point representation (i.e. is the point represented in affine or in projective coordinates). The projective coordinates have proven to be much more efficient than traditional affine coordinates because point additions and point doublings can be performed without field inversions which are the most time consuming operations. Mapping back to affine coordinates requires one inversion but still major speed-ups can be achieved by using projective coordinates in point representation instead of affine coordinates.

Efficient point multiplication algorithms have been developed because it is far too slow to compute Equation (30) by just adding point P to itself k times. The first and simplest efficient point multiplication algorithm is the binary method or add and double algorithm which relies on the binary representation of integer k. Certain efficient algorithms that require precomputations before the actual point multiplication have been developed and they are useful specially if specific curves and fixed base points are used. The fastest point multiplication method without precomputations seems to be Montgomery point multiplication in projective coordinates.

Fast and efficient ECC implementations on FPGAs have been published. The fastest implementation by Gura et al. performs point multiplication over a specific curve over $GF(2^{163})$ in 0.14 ms [12]. Major speedups (up to over 30 times) have been reported when hardware acceleration has been used instead of pure software implementation [18]. Overall, it can be said that FPGAs are a very inviting platform for ECC implementations because they provide the speed of hardware combined with the flexibity of software.

References

[1] G. B. Agnew, R. C. Mullin, and S. A. Vanstone. An Implementation of Elliptic Curve Cryptosystems Over $F_{2^{155}}$. *IEEE Journal on Selected Areas in Communications, Volume: 11 Issue: 5,* pages 804–813, June 1993.

[2] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, and J. Teich. Reconfigurable Implementation of Elliptic Curve Crypto Algorithms. *Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS'02, Marriott Marina, Fort Lauderdale, Florida, USA*, pages 157–164, April 15–19 2002.

[3] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography, London Mathematical Society Lecture Note Series 265.* Cambridge University Press, 2002.

[4] Dictionary of Scientific Biography. Vol. 5: Emil Fischer – Gottlieb Haberlandt, Charles Coulston Gillespie (editor in chief), Charles Scribner's Sons, New York, 1981.

[5] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory, Volume 22, Issue 6,* pages 644–654, November 1976.

[6] T. Elo. A Software Implementation of ECDSA on a Java Smart Card. Master's thesis, Helsinki University of Technology, Department of Computer Science, Telecommunications Software and Multimedia Laboratory, 2000.

[7] M. Ernst, M. Jung, F. Madlener, S. Huss, and R. Bluemel. A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over $GF(2^n)$. *Proceedings of Cryptographic Hardware and Embedded Systems, 4th International Workshop, CHES'02, Redwood Shores, CA, USA, LNCS 2523, Springer-Verlag,* pages 381–399, August 13–15 2002. B.S. Kaliski Jr., Ç.K. Koç, C. Paar (Eds.).

[8] Federal Information Processing Standards. Digital Signature Standard (DSS). *FIPS PUB 186-2,* January 27 2000. internet:
csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf, (May 20, 2003).

[9] Federal Information Processing Standards. Advanced Encryption Standard (AES). *FIPS PUB 197,* November 26 2001. internet:
csrc.nist.gov/publications/fips/fips197/fips-197.pdf, (May 20, 2003).

[10] Federal Information Processing Standards. Secure Hash Standard. *FIPS PUB 180-2,* August 1 2002. internet:
csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf, (May 20, 2003).

[11] J. B. Fraleigh. *A First Course in Abstract Algebra*. Addison-Wesley, 4th edition, 1989.

[12] N. Gura, H. Eberle, and S. C. Shantz. Generic Implementations of Elliptic Curve Cryptography using Partial Reduction. *Proceedings of the 9th ACM conference on Computer and communications security, CCS'02, Washington, DC, USA, ACM Press*, pages 108–116, November 18–22 2002.

[13] D. Hankerson, J. L. Hernandez, and A. Menezes. Software Implementation of Elliptic Curve Cryptography over Binary Fields. *Proceedings of Cryptographic Hardware and Embedded Systems, Second International Workshop, CHES'00, Worcester, MA, USA, LNCS 1965, Springer-Verlag Berlin Heidelberg*, pages 1–24, August 17–18 2000. Ç.K. Koç, C. Paar (Eds.).

[14] D. Johnson, A. Menezes, and S. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security, Volume 1, Issue 1, Springer-Verlag*, pages 36–63, 2001.

[15] T. Kerins, E. Popovici, W. Marnane, and P. Fitzpatrick. Fully Parametrizable Elliptic Curve Cryptography Processor over $GF(2^m)$. *Proceedings of the 12th International Conference on Field Programmable Logic and Applications, FPL 2002, Montpellier, France*, pages 750–759, September 2–4 2002. M. Glesner, P. Zipf, M. Renovell (Eds.).

[16] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation, Vol. 48*, pages 203–209, 1987.

[17] N. Koblitz. CM-Curves with Good Cryptographic Properties. *Advances in Cryptology - CRYPTO '91, Springer-Verlag Berlin Heidelberg*, pages 279–287, 1992. Joan Feigenbaum (Ed.).

[18] K. H. Leung, K. W. Ma, W. K. Wong, and P. H. W. Leong. FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor. *Proceedings of 2000 IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM'00, Napa Valley, California, USA*, pages 68–76, April 17–19 2000.

[19] J. López and R. Dahab. Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation. *Proceedings of Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, LNCS 1717, Springer-Verlag*, pages 316–327, August 1999. Çetin K. Koç, Christof Paar (Eds.).

[20] V. S. Miller. Use of Elliptic Curves in Cryptography. *Advances in Cryptology - CRYPTO '85, Lecture Notes in Computer Science, vol. 218, Springer-Verlag*, pages 417–426, 1985. Hugh C. Williams (Ed.).

[21] S. Okada, N. Torii, K. Itoh, and M. Takenaka. Implementation of Elliptic Curve Cryptography Coprocessor over $GF(2^m)$ on an FPGA. *Proceedings of Cryptographic Hardware and Embedded Systems, Second International Workshop, CHES'00, Worcester, MA, USA, LNCS 1965, Springer-Verlag Berlin Heidelberg*, pages 25–40, August 17–18 2000. Ç.K. Koç, C. Paar (Eds.).

[22] G. Orlando and C. Paar. A Super-Serial Galois Fields Multiplier for FPGAs and Its Application to Public-Key Algorithms. *Proceedings of Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '99, Napa Valley, California, USA*, pages 232–239, April 21–23 1999.

[23] G. Orlando and C. Paar. A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$. *Proceedings of Cryptographic Hardware and Embedded Systems, Second International Workshop, CHES'00, Worcester, MA, USA, LNCS 1965, Springer-Verlag Berlin Heidelberg*, pages 41–56, August 17–18 2000. Ç.K. Koç, C. Paar (Eds.).

[24] H. Pietiläinen. Elliptic Curve Cryptography on Smart Cards. Master's thesis, Helsinki University of Technology, Faculty of Information Technology, Department of Computer Science, 2000.

[25] Certicom Research. SEC 1: Elliptic Curve Cryptography. *Standards for Efficient Cryptography*, September 20 2000. Version 1.0, internet: www.secg.org/collateral/sec1_final.pdf, (May 28, 2003).

[26] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parametres. *Standards for Efficient Cryptography*, September 20 2000. Version 1.0, internet: www.secg.org/collateral/sec2_final.pdf, (May 28, 2003).

[27] M. Rosing. *Implementing Elliptic Curve Cryptography*. Manning Publications Co., 1999.

[28] M. Rosner. Elliptic Curve Cryptosystems on Reconfigurable Hardware. Master's thesis, Worcester Polytechnic Institute, 1998.

[29] RSA Security Inc. internet: http://www.rsasecurity.com, (May 28, 2003).

[30] B. Schneier. *Applied Cryptography, 2nd Edition*. John Wiley & Sons, Inc., 1996.

[31] J. H. Silverman. Fast Multiplication in Finite Fields $GF(2^N)$. *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems CHES'99, Worcester, Massachusetts, USA LNCS 1717, Springer-Verlag*, pages 122–134, August 1999. Çetin K. Koç, Christof Paar (Eds.).

[32] N. Smart, F. Hess, and P. Gaudry. Constructive and Destructive Facets of Weil Descent on Elliptic Curves. *HP Labs Technical Reports*, January 17 2000. internet: www.hpl.hp.com/techreports/2000/HPL-2000-10.html, (May 20, 2003).