

ECE 842 Report

Implementation of Elliptic Curve Cryptography

Wei-Yang Lin

December 15, 2004

Abstract

The aim of this report is to illustrate the issues in implementing a practical elliptic curve cryptographic system. Before doing the implementation, I will review group operation defined on elliptic curve over finite field. From that perspective, the efficiency of elliptic curve cryptographic system can be improved in two steps. The first step is to find a good representation of field element such that arithmetic over finite field can be done with less efforts. The other one is to find a good representation of point on elliptic curve such that time consuming field operation could be avoided. These two steps will be discussed in detail. Finally, we will implement a simple public key elliptic curve cryptographic system and show how computation time can be saved by applying these techniques.

1 Introduction

Elliptic curves over finite field can be used to implement secret exchanging scheme. This technique provides equivalent security level as RSA cryptography, but with shorter key length. Having shorter key length means smaller bandwidth and memory requirement. In some applications, these might be critical factors. Another advantage is that there are many curves available to be chosen. Consequently, the curve can be changed periodically for extra security.

The points on an elliptic curve form a group. The associated group operation involves arithmetic operations over underlying field. The field addition is easy to implement, both in hardware and in software. There has been much interest in the design of fast multiplication and fast inversion over $GF(2^n)$. In this report, I will discuss how group operation on elliptic curve can be accomplished efficiently.

The remainder of this report is organized as follows. Section 2 presents a brief summary of elliptic curve over finite fields. In section 3, we talk about using normal basis representation to perform arithmetic over finite field. Based on normal basis representation, we describe projective coordinates in section 4. Finally, experiment results show improvement achieved by applying normal basis and projective coordinates.

2 Elliptic curve over Galois fields

First, let's start with the general form of elliptic curves,

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Usually, they are categorized into following two versions,

$$y^2 + xy = x^3 + a_2x^2 + a_6 \tag{1}$$

or

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

The elliptic curves of the second form, Equation (2), is called *supersingular curve*. These curves have the advantage that they can be computed quickly. However, because of their special properties, their corresponding discrete logarithm problem is relatively easy to solve. That makes supersingular curves unsuitable for cryptographical purpose.

Equation (1) is called *non-supersingular curve*. To date, no attacking method takes less than exponential time. Therefore, they are excellent for cryptography.

The points on elliptic curve together with a point at infinite form a group. For elliptic curves over real number, the associated group operation can be visualized in terms of their geometrical relationship [5]. To implement a cryptographical system, we will focus on the case where elliptic curve over $GF(2^n)$. It leads us to efficient software and hard ware implementation.

Unfortunately, the group operation over $GF(2^n)$ is hard to visualize. It also require a lot of algebra to derive its formula. Here, I will skip all the derivation and only state the results. Based on these results, it will be very easy to understand the implementation issues.

Given two points, P and Q , on a elliptic curve, group operation is defined as

$$P + Q = R$$

Let (x_1, y_1) , (x_2, y_2) and (x_3, y_3) denote the coordinates of P , Q and R respectively. Then, x_3, y_3 can be expressed in terms of x_1, y_1, x_2 and y_2 as below: if $P \neq Q$:

$$\theta = \frac{y_2 - y_1}{x_2 - x_1}$$

$$x_3 = \theta^2 + \theta + x_1 + x_2 + a_2$$

$$y_3 = \theta(x_1 + x_3) + x_3 + y_1$$

if $P = Q$:

$$\theta = x_1 + \frac{y_1}{x_1}$$

$$x_3 = \theta^2 + \theta + a_2$$

$$y_3 = x_1^2 + (\theta + 1)x_3$$

They are called *point addition* and *point doubling* on elliptic curve. In the elliptic curve cryptographic system, we will send *multiplication of point* over a unsecured channel. Multiplication of point can be computed efficiently using only point addition and point doubling. For example,

$$15P = P + 2(P + 2(P + 2P))$$

Now, it should be very clear that *point doubling* and *point addition* are the fundamental operations in elliptic curve cryptographic system. These two operations involve multiplication, division, addition, subtraction and squaring over $GF(2^n)$. So, our goal is to implement the arithmetics over $GF(2^n)$ with minimal hardware or software complexity.

3 Arithmetics over $GF(2^n)$

$GF(2^n)$ can be viewed as vector space of dimension m over $GF(2)$. That is, there exist a basis $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$ such that each $a \in GF(2^n)$ can be written uniquely in the form

$$a = \sum_{i=0}^{n-1} a_i \alpha_i, \text{ where } a_i \in \{0, 1\}$$

We can also represent a as a vector $(a_0, a_1, \dots, a_{n-1})$. In hardware, a field element is stored in an array of length n . Addition of field element is accomplished by bitwise XOR operation. Because vector space is over $GF(2)$, subtraction of field elements is exactly the same as addition of field elements.

3.1 Normal Basis

A normal basis in $GF(p^n)$ is a basis of the form $\{\beta^{p^0}, \beta^{p^1}, \beta^{p^2}, \dots, \beta^{p^{n-1}}\}$. It is well known that a normal basis exists in every finite field. Every $B \in GF(p^n)$ may be uniquely expressed in terms of

$$B = \sum_{i=0}^{n-1} b_i \beta^{p^i}$$

Further, let

$$A = \sum_{i=0}^{n-1} a_i \beta^{p^i}, C = \sum_{i=0}^{n-1} c_i \beta^{p^i}$$

and

$$C = AB$$

Now

$$C = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \beta^{p^i} \beta^{p^j}$$

Since we can write

$$\beta^{p^i} \beta^{p^j} = \sum_{k=0}^{n-1} \lambda_{i,j}^{(k)} \beta^{p^k} \tag{3}$$

Substitution yields

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \lambda_{i,j}^{(k)} \quad (4)$$

Define a matrix \mathbf{T}_N as follows: Index the rows of \mathbf{T}_N by the ordered pairs (i, j) . In row (i, j) , column k put $\lambda_{i,j}^{(k)}$.

From now on, we consider the special case where $p = 2$. This leads us to design an efficient implementation. Let $A = (a_{n-1}, \dots, a_1, a_0)$ denote the coordinate vector for A in the normal basis. Because the property of normal basis, we have

$$\begin{aligned} A^2 &= (a_{n-2}, \dots, a_0, a_{n-1}) \\ A^{2^m} &= (a_{n-m-1}, \dots, a_{n-m+1}, a_{n-m}) \end{aligned}$$

where the subscripts are taken modulo n . This is an amazing result because square of field element can be calculated by circular shifting. Many implementation techniques are based on exploiting this property.

By raising both side of (3) to the power of 2^{-m} , then

$$(\beta^{2^i} \beta^{2^j})^{2^{-m}} = \beta^{2^{i-m}} \beta^{2^{j-m}} = \sum_{k=0}^{n-1} \lambda_{i-m, j-m}^{(k)} \beta^{2^k} = \sum_{k=0}^{n-1} \lambda_{i,j}^{(k)} \beta^{2^{k-m}}$$

Equating the coefficients of β^{2^0} in the above equation, yields

$$\lambda_{i-m, j-m}^{(0)} = \lambda_{i,j}^{(m)} \quad (5)$$

Therefore Equation (4) can be written as

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \lambda_{i-k, j-k}^{(0)} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{i+k} b_{j+k} \lambda_{i,j}^{(0)}$$

Therefore, c_k is obtained from m -fold cyclic shift of the variables involved.

3.1.1 Optimal normal basis

An *Optimal Normal Basis* (ONB) [4] is one with the minimum number of nonzero terms in Equation (4). This value was proved to be $2n - 1$ [1]. Since

it allows multiplication with minimum complexity, such a basis normally leads to a more efficient hardware implementation.

There are two types of optimal normal bases [4], namely Type I and Type II. In the following sections, we show the conditions under which optimal normal basis exist and the procedures for constructing multiplication table.

3.1.2 Type I

For Type I, there exists an optimal normal basis if

1. $n + 1$ is a prime
2. 2 is a primitive in $GF(n + 1)$

To implement multiplication, we need to find $\lambda_{i,j}^{(k)}$ in Equation (4). According to Equation (5), we only need to solve for $\lambda_{i,j}^{(0)}$ and then the whole multiplication table can be derived from $\lambda_{i,j}^{(0)}$. For type I ONB, $\lambda_{i,j}^{(0)} = 1$ iff i, j satisfy one of the following two conditions [5],

$$2^i + 2^j = 1 \pmod{n + 1}$$

$$2^i + 2^j = 0 \pmod{n + 1}$$

Using these two condition and Equation (5), building multiplication table is simple and efficient. The multiplication table of $GF(2^4)$ is shown in Table 1.

3.1.3 Type II

For Type II, an optimal normal basis exists if

1. 2 is primitive in \mathbb{Z}_{2n+1} , or
2. $2n+1$ is a prime congruent to 3 modulo 4 and 2 generates the quadratic residues in \mathbb{Z}_{2n+1} .

For type II ONB, $\lambda_{i,j}^0 = 1$ iff i, j satisfy one of the following conditions [5],

$$2^i + 2^j = +1 \pmod{2n + 1}$$

$$2^i + 2^j = -1 \pmod{2n + 1}$$

$$2^i - 2^j = +1 \pmod{2n + 1}$$

$$2^i - 2^j = -1 \pmod{2n + 1}$$

Table 1: Multiplication table of Type I ONB, $GF(2^4)$

i	j	k			
		0	1	2	3
0	0	0	1	0	0
0	1	0	0	0	1
0	2	1	1	1	1
0	3	0	0	1	0
1	0	0	0	0	1
1	1	0	0	1	0
1	2	1	0	0	0
1	3	1	1	1	1
2	0	1	1	1	1
2	1	1	0	0	0
2	2	0	0	0	1
2	3	0	1	0	0
3	0	0	0	1	0
3	1	1	1	1	1
3	2	0	1	0	0
3	3	1	0	0	0

Again, using these conditions and Equation (5), building the multiplication table is straight forward.

3.2 Fast Inversion

Using normal basis representation, the square of a field element is very easy to calculate. That is just a rotation. Based on this nice property, Itoh and Tsujii [2] proposed an efficient method for computing the inverse of a field element.

Observe that if $a \in GF(2^n)$, $a \neq 0$, then

$$a^{-1} = a^{2^n - 2}$$

This is the result from Fermat's Theorem. Now, let's look at the exponent. It can be factored as

$$2^n - 2 = 2(2^{n-1} - 1)$$

That means we can calculate $a^{(2^{n-1}-1)}$ first and then square it. Then, if m is odd, we have

$$2^{n-1} - 1 = (2^{(n-1)/2} + 1)(2^{(n-1)/2} - 1)$$

Therefore, it takes only one multiplication once $a^{2^{(n-1)/2}-1}$ has been computed. If n is even, we have

$$2^{n-1} - 1 = 2(2^{(n-2)/2} + 1)(2^{(n-2)/2} - 1) + 1$$

and consequently it takes two multiplications once $a^{2^{(n-2)/2}-1}$ has been computed. The procedure is then repeated recursively.

It can be shown that this method requires $\lfloor \log_2(n-1) \rfloor + \omega(n-1) - 1$ field multiplications, where $\omega(n-1)$ denotes number of 1's in the binary representation of $m-1$.

4 Projective Coordinates

Even though there are special techniques for computing inverse in $GF(2^n)$, a field inversion is still far more expensive than a field multiplication. It may be of computational advantage to avoid inversion. This is done with the use of *projective coordinates* [3].

4.1 Basic facts

A *projective coordinate* is defined such that (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) are said to be equal if they are related by $X_1 = \lambda X_2, Y_1 = \lambda Y_2, Z_1 = \lambda Z_2, \lambda \neq 0$. Note that if a point $P = (X, Y, Z)$ has nonzero Z , then P can be represented by the projective point $(x, y, 1)$, where $x = X/Z$ and $y = Y/Z$.

An equation $f(x, y) = 0$ corresponds to an equation $F(X, Y, Z) = 0$, where F is obtained by replacing $x = X/Z, y = Y/Z$, and multiplying with a power of Z to clear the denominators. In particular, the *projective equation* of $y^2 + xy = x^3 + ax^2 + b$ is given by

$$ZY^2 + XYZ = X^3 + aX^2Z + bZ^3$$

If $Z = 0$ in this equation, then $X^3 = 0$, i.e., $X = 0$. Therefore, $(0, 1, 0)$ is the only projective point that satisfies this equation. This point is called *the point at infinity* and denoted as \mathcal{O} .

To convert a point (x, y) to projective coordinate, one sets $X = x \cdot Z, Y = y \cdot Z, Z = Z$. For elliptic curve over $GF(2^n)$, the rule to negate a point $P = (x, y)$ is

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

So, the projective coordinate of $-P$ is given by $(X, X + Y, Z)$.

Let $P_0 = (X_0, Y_0, Z_0)$ and $P_1 = (X_1, Y_1, 1)$ be two different points on elliptic curve and assume that $P_1 \neq \mathcal{O}$ and $P_1 \neq -P_0$. The adding formula in projective coordinate is

$$(X_0, Y_0, Z_0) + (X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$$

$$\begin{aligned} X_2 &= AD \\ Y_2 &= CD + A^2(BX_0 + AY_0) \\ Z_2 &= A^3Z_0 \end{aligned}$$

and where

$$\begin{aligned} A &= X_1Z_0 + X_0 \\ B &= Y_1Z_0 + Y_0 \\ C &= A + B \\ D &= A^2(A + aZ_0) + Z_0BC \end{aligned}$$

This addition can be done in 13 field multiplications, which is more than the 2 field multiplications required when using affine coordinates. However, we usually save computation time by not performing field inversion. The gain occurs at the expense of memory space, as we now need extra registers to store intermediate results.

The doubling formula in projective coordinate is

$$(X_0, Y_0, Z_0) + (X_0, Y_0, Z_0) = (X_2, Y_2, Z_2)$$

$$\begin{aligned} X_2 &= A \cdot B \\ Y_2 &= X_1^4A + B(X_1^2 + Y_1Z_1 + A) \\ Z_2 &= A^3 \end{aligned}$$

where

$$\begin{aligned} A &= X_1 Z_1 \\ B &= bZ_1^4 + X_1^4 \end{aligned}$$

Therefore, point doubling can be done in 7 multiplications, which again is usually an improvement on the formulae with affine coordinates which needs 1 inversion and 2 multiplications.

4.2 Improved projective coordinate

In [3], the authors derived the new formula for adding points in projective coordinates. The *improved projective coordinate* is defined such that (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) are said to be equal if they are related by $X_1 = \lambda X_2, Y_1 = \lambda^2 Y_2, Z_1 = \lambda Z_2, \lambda \neq 0$.

An equation $f(x, y) = 0$ corresponds to an equation $F(X, Y, Z) = 0$, where F is obtained by replacing $x = X/Z, y = Y/Z^2$, and multiplying with a power of Z to clear the denominators. In particular, the *projective equation* of $y^2 + xy = x^3 + ax^2 + b$ is given by

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$$

If $Z = 0$ in this equation, then $Y^2 = 0$, i.e., $Y = 0$. Therefore, $(1, 0, 0)$ is the only projective point that satisfies this equation. Hence $\mathcal{O} = (1, 0, 0)$ is *the point at infinity* when using improved projective coordinates.

Let $P_0 = (X_0, Y_0, Z_0)$ and $P_1 = (X_1, Y_1, Z_1)$ be two different points on elliptic curve. The adding formula in improved projective coordinate is

$$(X_0, Y_0, Z_0) + (X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$$

$$\begin{aligned} X_2 &= C^2 + H + G \\ Y_2 &= H \cdot I + Z_2 \cdot J \\ Z_2 &= F^2 \end{aligned}$$

where

$$\begin{aligned} A_0 &= Y_1 \cdot Z_0^2, & C &= A_0 + A_1, & G &= D^2 \cdot (F + aE^2), \\ A_1 &= Y_0 \cdot Z_1^2, & D &= B_0 + B_1, & H &= C \cdot F, \\ B_0 &= X_1 \cdot Z_0, & E &= Z_0 \cdot Z_1, & I &= D^2 \cdot B_0 \cdot E + X_2, \\ B_1 &= X_0 \cdot Z_1, & F &= D \cdot E, & J &= D^2 \cdot A_0 \cdot E + X_2, \end{aligned}$$

During the computation of point multiplication, we keep doing point doubling and also adding base point when it is necessary. The Z coordinate of base point is 1. Hence we are more interested in this special case. The point adding formula for $Z_1 = 1$ is given by:

$$(X_0, Y_0, Z_0) + (X_1, Y_1, 1) = (X_2, Y_2, Z_2)$$

$$\begin{aligned} X_2 &= A^2 + D + E \\ Y_2 &= E \cdot F + Z_2 \cdot G \\ Z_2 &= C^2 \end{aligned}$$

where

$$\begin{aligned} A &= Y_1 \cdot Z_0^2 + Y_0, & E &= A \cdot C, \\ B &= X_1 \cdot Z_0 + X_0, & F &= X_2 + X_1 \cdot Z_2, \\ C &= Z_0 \cdot B, & G &= X_2 + Y_1 \cdot Z_2, \\ D &= B^2 \cdot (C + aZ_0^2), \end{aligned}$$

Point addition in improved projective coordinate can be done in 13 field multiplications. If $Z_1 = 1$, then only 10 field multiplications are required which is less than the field multiplications required when using original projective coordinates. However, the gain occurs at the expense of using more memory space.

The doubling formula in improved projective coordinate is

$$(X_1, Y_1, Z_1) + (X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$$

where

$$\begin{aligned} Z_2 &= Z_1^2 \cdot X_1^2, \\ X_2 &= X_1^4 + b \cdot Z_1^4, \\ Y_2 &= b \cdot Z_1^4 \cdot Z_2 + X_2 \cdot (a \cdot Z_2 + Y_1^2 + b \cdot Z_1^4). \end{aligned}$$

The improved projective doubling algorithm requires 5 field multiplications. Since it takes 2 field multiplications less than the previous projective doubling algorithm, we obtain an improvement of about 30% in general. The number of multiplications and squarings required to perform an elliptic group operation for various kinds of projective coo is listed in Table 2.

Table 2: The number of operations for different kinds of projective coordinates

Projective coordinates	Doubling		Adding		Cost of $2^5P + Q$	
	#Mult.	#Sqr.	#Mult.	#Sqr.	#Mult.	#Sqr.
$(x/z, y/z^2)$	4	5	9	4	29	29
$(x/z^2, y/z^3)$	5	5	10	4	35	29
$(x/z, y/z)$	7	5	12	1	47	26

5 Software Implementation

The running time obtained with software implementation is presented in this section. The platform consisted of a $2.8GHz$ Pentium processor, windows XP and visual C++ 6.0.

As in most texts on cryptography, we said Alice wants to share secret with Bob. Diffie-Hellman scheme can be described as follows. Let k_a be Alice's private key and k_b belongs to Bob. Alice compute public key P_a and send it to bob.

$$P_a = k_a B$$

where B is base point. Bob also compute public key P_b and send it to Alice.

$$P_b = k_b B$$

Then, they both compute the shared secret P_s .

$$P_s = k_a(k_b B) = k_b(k_a B)$$

We use the running time required to do all these computation, specifically computing public keys and share secret, as a metric to compare different implementation techniques.

There are many useful software routines for elliptic curve cryptography in [5]. Diffie-Hellman scheme is built based on these routines. Then, running times are measured under following conditions:

- Use *Optimal Normal Basis* as representation of field elements.
- Use *Optimal Normal Basis* as representation of field elements and *projective coordinate* as representation of point on a elliptic curve.

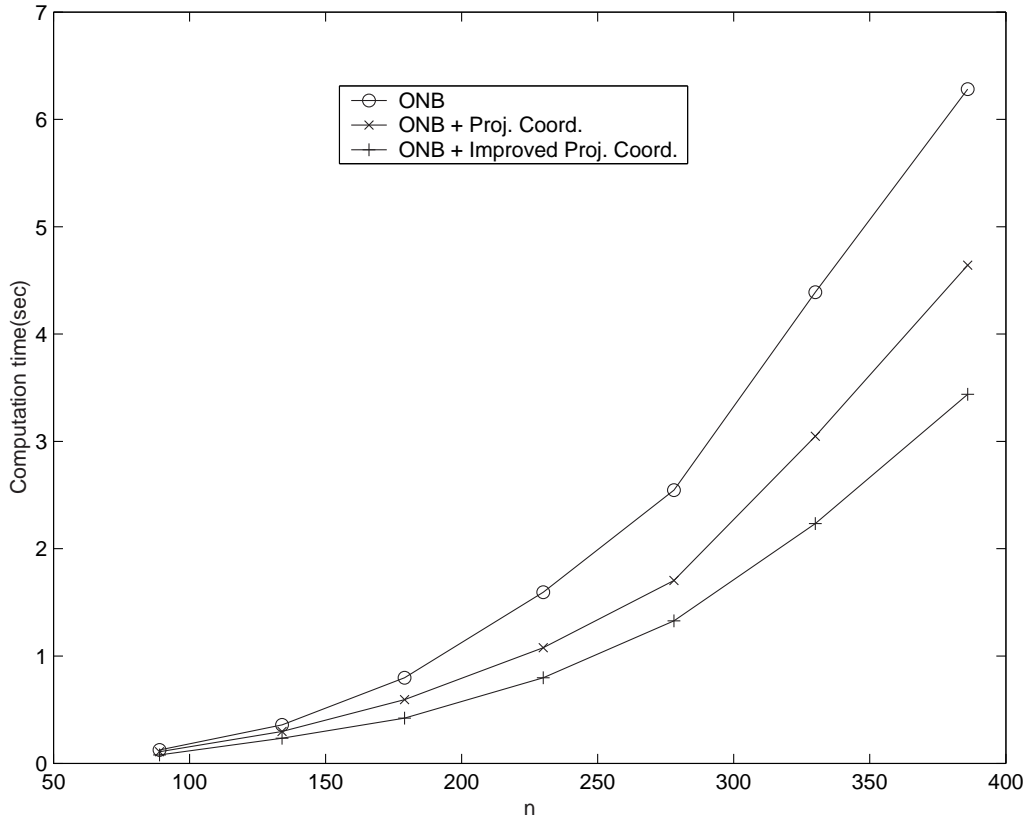


Figure 1: Running time for $n = 89, 134, 179, 230, 278, 330, 386$. ONB stands for *Optimal Normal Basis*.

- Use *Optimal Normal Basis* as representation of field elements and *improved projective coordinate* as representation of point on a elliptic curve.

Here, we choose the finite fields $GF(2^n)$ where optimal normal basis exist. The resulting running times are shown in Figure 1. By using projective coordinate, the time consuming field inversion can be avoided. Hence running time is reduced as we can see in Figure 1. We can further reduce running time by applying *improved projective coordinate*. The improvement can also be observed in Figure 1. The results are what we expect from theoretical point of view.

6 Conclusion

Sometimes, implementation is not a trivial task even though you know the mathematics very well. For example, implementation of *Fast Fourier Transform* in C language is not so easy for most of the engineering students. Implementation of elliptic curve cryptography is a similar situation. Hence, finding some software routines to begin with will save you lots of time. For those who are interested in implementing elliptic curve cryptography in computer, [5] is an excellent starting point. It provides complete set of software routines for elliptic curve cryptography. Based on these routines, it is very easy to implement advanced techniques and see how it works. In this report, I implement Diffie-Hellman scheme and use it as test platform for different implementation techniques. The results give us direct comparison on performance of different implementation techniques.

References

- [1] D. W. Ash, I. F. Blake, and S. A. Vanstone. Low complexity normal bases. *Discrete Applied Mathematics*, 25(3):191–210, Nov 1989.
- [2] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Information and Computation*, 78(3):171–177, SEP 1988.
- [3] J. Lopez and R. Dahab. *Improved algorithms for elliptic curve arithmetic in $GF(2^n)$* , pages 201–12. Selected Areas in Cryptography. 5th Annual International Workshop, SAC'98. Proceedings. Springer-Verlag, Kingston, Ont., Canada, 1999.
- [4] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson. Optimal normal bases in $GF(p^n)$. *Discrete Applied Mathematics*, 22(2):149–61, Feb 1989.
- [5] Michael Rosing. *Implementing Elliptic Curve Cryptography*. Manning Publications, 1998.