New Table Look-up Methods for Faster Frobenius Map Based Scalar Multiplication Over $GF(p^n)$

Palash Sarkar, Pradeep Kumar Mishra and Rana Barua.

Cryptology Research Group,
Indian Statistical Institute, 203 B T Road,
Kolkata-700108, INDIA

Abstract. We describe a new scalar multiplication algorithm for elliptic and hyperelliptic curve cryptosystems. The algorithm is obtained by combining Koblitz's idea of using Frobenius automorphism along with a very special kind of look-up table. In the case where the base point is unknown, we present an efficient algorithm to compute the look-up table online. Our algorithm applies to prime power fields $GF(p^n)$. One important subclass of such fields are Optimal Extension Fields (OEF's) which are believed to be ideal for efficient implementation of cryptographic primitives. Over prime power fields, our algorithm compares favourably to other known algorithms for scalar multiplication.
Keywords : scalar multiplication, Frobenius map, elliptic curves, hyperelliptic curves, window methods, look-up table, normal basis.

1 Introduction

Elliptic and hyperelliptic curves provide a rich source of cyclic groups over which the discrete logarithm problem is believed to be hard. Hence these groups are suitable for defining public key cryptosystems. The dominant operation in any such cryptosystem is the so called *scalar multiplication*, which is the operation of computing mX, where m is an integer and X is either a point of an elliptic curve or a reduced divisor in the Jacobian of a hyperelliptic curve.

The efficiency of an elliptic or hyperelliptic curve cryptosystem is crucially dependent on the speed of scalar multiplication. Not surprisingly, this has led to a tremendous research in algorithms for fast scalar multiplication. These algorithms fall naturally into two classes.

- General algorithms which work for any cyclic group.
- Algorithms which exploit the algebraic properties of elliptic and hyperelliptic curves.

One of the most important technique of the second kind is the use of endomorphisms to speed up scalar multiplication. This was first proposed by Koblitz [13] and has also been studied by later authors (for example see [4, 5, 7, 10, 14, 23, 24]). The most natural endomorphism is the Frobenius automorphism and was initially proposed by Koblitz [13]. A series of research papers have resulted in the applicability of the Frobenius map technique to elliptic and hyperelliptic curves over any finite field. See [14] for a more detailed description of the development of this technique.

Let F_q be the underlying field. The Frobenius map technique (and hence our algorithm) really applies when the q is a prime power (rather than a prime). The case $p = 2$ has been explored extensively by the researcher community. Our algorithms apply to the case $p > 2$, for example, for Optimal Extension Fields. Optimal Extension Fields (OEF's) are finite fields of the form $GF(p^m),\ p > 2$, where p and m are chosen to match the underlying hardware. OEF's, optimally utilising the underlying hardware offer considerable advantage in software implementations of elliptic curve cryptosystems. In prime power fields, of which OEF's are special cases, our algorithm provides a substantial reduction in the number of point arithmetic (addition/doubling) operations as compared to other existing algorithms.

In this work, we concentrate on developing a new scalar multiplication algorithm based on the Frobenius map. The basic idea of the algorithm is known and has been described for both elliptic curves [10, 24, 23] and hyperelliptic curves [4, 14]. The principle innovation that we introduce is a very special kind of look-up table. Given a point X, we define a look-up table Tab_X in the following manner: The table stores 3^h points and for $(a_0, \ldots, a_{h-1}) \in \{0, \pm 1\}^h$, we define $\mathsf{Tab}_X[a_0, \ldots, a_{h-1}]$ to be the point

$$\mathsf{Tab}_X[a_0, \ldots, a_{h-1}] = a_0 X + a_1 \phi(X) + \cdots + a_{h-1}\phi^{h-1}(X) \qquad (1)$$

where ϕ is the Frobenius map. This is a simple idea. Aoki et al [1] have worked on similar lines for elliptic curves. A proper utilization of this idea provides a substantial reduction in the number of point arithmetic operations.We also extend the idea to the situation when $\{a_0, \ldots, a_{h-1}\} \in \{0, \pm 1, \pm 2, \ldots, \pm 2^w\}^h$ for some $w > 0$. The table can be precomputed and stored when the point X is known in advance. However, there are applications where the point X is not known in advance. We present an algorithm to compute Tab_X in such a situation. It turns out that by using a simple trick, it is possible to reduce the number of point additions needed to compute Tab_X.

The size of the look-up table is determined by the number of points to be stored. In the case, where the underlying field is represented using normal basis, the number of points required to be stored is quite small. However, if polynomial basis representation of the field elements are used, then the storage requirement increases. Thus our algorithm is most useful when the underlying field is represented using normal basis.

The plan of the paper is as follows. In Section 2, we present the necessary preliminaries. Section 3 describes the basic idea of the table look-up algorithm. This is developed into a general look-up table based algorithm to compute scalar multiplication in Section 4. In Section 5, we describe algorithms to compute the look-up tables online. Section 6presents a detailed discussion on the results obtained and compares the performance of the algorithm to other scalar multiplication algorithms. Finally, Section 7 concludes the paper.

2 Preliminaries

Let K be a field and $\overline{K}$ be the algebraic closure of K. A *hyperelliptic curve* C of genus g over K is an equation of the form $C : v^2 + h(u)v = f(u)$ where $h(u)$ in $K[u]$ is a polynomial of degree at most g, $f(u)$ in $K[u]$ is a monic polynomial of degree $2g + 1$, and there are no solutions (u, v) in $\overline{K} \times \overline{K}$, which simultaneously satisfy the equations

$$\left.\begin{array}{ll} v^2 + h(u)v & = f(u); \\ 2v + h(u) & = 0; \\ h'(u)v - f'(u) = 0. & \end{array}\right\} \tag{2}$$

Elliptic curves are hyperelliptic curves of genus 1. If L is any extension field of K, then the set of all L-rational points of C is the set $\{(x, y) \in L \times L : y^2 + h(x)y = f(x)\} \bigcup \{\infty\}$ where ∞ is a special point called the point at infinity. The set of L-rational points of an elliptic curve form a group under a suitably defined addition operation. On the other hand, for $g > 1$, the set of points on a hyperelliptic curve does not form a group. Instead it is customary to consider the free abelian group generated by the set of points. Elements of this group are called divisors. The set of certain special kinds of divisors called *reduced divisors* form an additive group.

The additive group of the set of points of an elliptic curve has been used to obtain ElGamal type cryptosystems. Similarly, the group of reduced divisors of hyperelliptic curves has also been proposed for such types of cryptosystems [11]. One of the most important structures for practical applications is the binary Koblitz curves, which are elliptic curves over the binary field.

In the rest of the paper by a *point* we will mean either a point on an elliptic curve or a reduced divisor of an hyperelliptic curve. The main operation for realizing elliptic and hyperelliptic curve cryptosystems is mX, where m is an integer and X is a point. This operation is called *scalar multiplication*. Our focus in this paper will be to obtain efficient algorithms for scalar multiplication.For details of elliptic and hyperelliptic curve cryptosystems we refer the reader to [11, 17, 3].

2.1 Prime Power Fields

For cryptographic applications, binary fields $GF(2^n)$ and prime fields $GF(p)$ were considered most attractive for software implementations. Later Optimal Extension Fields (OEF's) [2], a special class of finite fields of the form $GF(p^m)$ were proposed, where p and m were chosen suitably to exploit the underlying hardware optimally for performance gain. An OEF is a finite field of the form $GF(p^n)$, where (i) p is a pseudo-Mersenne prime and (ii) an irreducible binomial $P(x) = x^n - \omega$ exists over $GF(p)$. The prime p is generally chosen to be very close to the word size of the processor, so that each machine word can accomodate one element of the subfield $GF(p)$ and each element of the OEF $GF(p^n)$, can be accomodated in n words, with minimum wastage of memory. Also, OEF's allow efficient modular reduction for arithmetic in the extension field. The algorithms proposed in this work are suitable for the prime power fields of type $GF(p^n)$, which contains the OEF's as a subclass of it.

2.2 Normal Basis

Let q be a prime power. A field F_{q^n} is said to have a normal basis if it has a basis (over F_q) of the form $\{\alpha, \alpha^q, \cdots, \alpha^{q^{n-1}}\}$. Any element of the field can be represented as $x = \sum_{j=0}^{n-1} a_j \alpha^{q^j}$ or briefly as an ordered n-tuple $x = (a_0, \cdots, a_{n-1})$. In the field F_{q^n}, we have, $x^q = (\sum_{j=0}^{n-1} a_j \alpha^{q^j})^q = (\sum_{j=0}^{n-1} a_j \alpha^{q^{j+1}})$. Thus if x is represented by the tuple $(a_0, \cdots, a_{n-1})$ then x^q is represented by $(a_{n-1}, a_0, \cdots, a_{n-2})$, as $\alpha^{q^n} = 1$. With a normal basis representation of elements, x^q can be computed from x by a circular shift operation only. See [16] for more details on normal basis.

2.3 Frobenius Map

Let F_q be a finite field. The Frobenius map $\phi : F_{q^n} \to F_{q^n}$ is an automorphism of F_{q^n} and is defined as $\phi(x) = x^q$. The map is extended to points of an elliptic or hyperelliptic curve over F_{q^n} in the following manner: A point of an elliptic curve is represented using a pair of elements of F_{q^n}; similarly a reduced divisor of a hyperelliptic curve is represented using a tuple of elements of F_{q^n}. An application of the Frobenius map to a point is to actually apply the map individually to the field elements which represent the point. We note that ϕ^n is the identity map on F_{q^n}. If the field F_{q^n} is represented using a normal basis, then the computation of $\phi(x)$ is "for free". Further, as observed in [24, 23], in the case $q = 2$, the Frobenius map is $\phi(x) = x^2$ and hence can be computed using a field squaring which is a relatively cheap operation even if polynomial basis representation of elements is used.

2.4 Scalar Multiplication using Frobenius Map

In [13], Koblitz had suggested the use of Frobenius map to speed up scalar multiplication algorithm. This idea has later been developed by several authors [5, 7, 10, 18, 22, 23]. For hyperelliptic curves, it has been shown [14, 4] that the Frobenius map based method can be used over any field of finite characteristic.

Let q be a prime power, F_q be the finite field of order q and F_{q^n} an extension field of F_q. Let C be the curve of genus g to be used for the cryptosystem and we consider the F_{q^n}-rational points of C. Let ϕ be the Frobenius map from F_{q^n} to F_{q^n}. Let m be an integer, X a point (either a point of an elliptic curve or a reduced divisor of a hyperelliptic curve) and we wish to compute mX. The base-ϕ expansion of m is $\sum_{i=0}^{n-1} u_i \phi^i$, where under reasonable assumptions each u_i is an integer in the range $[-q^g, q^g]$. It is possible to obtain the base-ϕ expansion of m. Next we define some additional parameters which will be required in the rest of the paper.

1. $A = \max\lfloor \log_2(|u_i|)\rfloor$.
2. For $i \in \{0, \ldots, n-1\}$ write $|u_i| = \sum_{j=0}^{A} u'_{i,j} 2^i$, where $u'_{i,j} \in \{0, 1\}$.
3. $u_{i,j} = \mathsf{sgn}(u_i) u'_{i,j}$, where $\mathsf{sgn}(u_i)$ is the sign of u_i.
4. For $0 \leq i \leq n-1$, define $X_0 = X$ and $X_i = \phi^i(X_0) = \phi^i(X)$.
5. Parameters h and w are respectively the column and row window sizes.
6. Parameters s and r are defined by the equation:
   $n = s \times h + r$, where r is a unique integer in the set $\{1, \ldots, h\}$.
6. Parameter $k = \lceil (A+1)/w \rceil$.

The expression mX can be written as

$$\left.\begin{aligned} mX &= u_0 X_0 + u_1 X_1 + \cdots + u_{n-1} X_{n-1} \\ &= (u_{0,0} + u_{0,1} 2 + \cdots + u_{0,A} 2^A) X_0 \\ &\quad + (u_{1,0} + u_{1,1} 2 + \cdots + u_{1,A} 2^A) X_1 \\ &\quad + \cdots \\ &\quad + (u_{n-1,0} + u_{n-1,1} 2 + \cdots + u_{n-1,A} 2^A) X_{n-1} \end{aligned}\right\} \tag{3}$$

We consider the above expression to be an $n \times (A+1)$ matrix. Let $\tau = q^n$. Then depending on the nature of the underlying field F_τ, there are several cases.

1. Case $n = 1$ and $\tau = q$ is a prime: In this case, (3) reduces to a single row. In this situation, the Frobenius map based technique does not really apply. Hence we will not consider this kind of fields in this paper.
2. Case $n > 1$: In this situation (3) will have more than one rows and the Frobenius map technique can be applied. It will be convenient to divide this into two subcases.
   - Subcase $q = 2$: The field is F_{2^n} and the curves are the binary Koblitz curves. In this case each $u_i \in \{0, \pm 1\}$ and hence (3) is actually a single column. This is the other extreme to Case 1 above. In [24, 23], equation (3) is called the ϕ-adic expansion of m.
   - Subcase $q > 2$: In this situation, (3) has a more square shape and again our algorithm offers improvements over existing algorithms.

The following simple algorithm can be used to compute mX from (3) (see [4, 10, 14, 23, 24]).

Algorithm 1.
Input : integer $m = \sum_{i=0}^{n-1} u_i \phi^i$ and point X .
Output : mX.

1. For $0 \leq i \leq n-1$ and $0 \leq j \leq A$, compute X_i and $u_{i,j}$;
2. Set $Y = \sum_{i=0}^{n-1} u_{i,A} X_i$;
3. For $j = A-1$ down to 0
4. $\quad\quad Y = 2Y$; $Y = Y + \sum_{i=0}^{n-1} u_{i,j} X_i$
5. return Y.

Proposition 1. *In the above algorithm, the average numbers of additions and doublings needed to compute mX are $n(A+1)/2$ and A respectively.*

3 Basic Table Look-Up Methods

We describe a new table look-up method to compute mX from (3). We observe that the right hand side of (3) has the structure of a matrix. Algorithm 1 performs a column by column computation. Our first observation is the fact that the number of rows in (3) is equal to n (the extension degree of F_{q^n} over F_q) and is independent of both m and X. Given a point X, we define a table Tab_X in the following manner: There are 3^n entries in Tab_X which are indexed by the elements of $\{0, \pm 1\}^n$. For any $(b_0, \ldots, b_{n-1}) \in \{0, \pm 1\}^n$, we define

$$\mathsf{Tab}_X[b_0, \ldots, b_{n-1}] = b_0 X_0 + \cdots + b_{n-1} X_{n-1} = b_0 X + b_1 \phi(X) + \cdots + b_{n-1} \phi^{n-1}(X) \quad (4)$$

Hence the look-up table Tab_X stores 3^n points. If this table is available, then computing mX becomes quite easy and is described by the following algorithm.

Algorithm 2.
Input : $m = \sum_{i=0}^{n-1} u_i \phi^i$ and point X .
Output : mX.
1. Set $Y = \mathsf{Tab}_X[u_{0,A}, \ldots, u_{n-1,A}]$
2. For $j = A - 1$ down to 0
3. $\quad Y = 2Y$;
4. $\quad Y = Y + \mathsf{Tab}_X[u_{0,j}, \ldots, u_{n-1,j}]$;
5. return Y.

Proposition 2. *Algorithm 2 computes mX using A additions and A doublings. The table Tab_X stores 3^n points.*

3.1 Using Smaller Look-up Tables

In Algorithm 2 we use a table of 3^n points, where n is the extension degree of F_{q^n} over F_q. If n is relatively small (≤ 4), then the table is of moderate size. However, if n is larger, then the required storage space may be prohibitively high. In this section, we show how to tackle this problem.

Let h ($1 \leq h \leq n$) be a small positive integer which is the column window size. Write $n = s \times h + r$, where r is a unique integer from $\{1, \ldots, h\}$. Then for $(a_0, \ldots, a_{n-1}) \in \{0, \pm 1\}^n$ we can write

$$\begin{aligned} a_0 X_0 + a_1 X_1 + \cdots + a_{n-1} X_{n-1} &= a_0 X_0 + a_1 X_1 + \cdots + a_{h-1} X_{h-1} \\ &\quad + a_h X_h + a_{h+1} X_{h+1} + \cdots + a_{2h-1} X_{2h-1} \\ &\quad \vdots \\ &\quad + a_{(s-1)h} X_{(s-1)h} + \cdots + a_{sh-1} X_{sh-1} \\ &\quad + a_{sh} X_{sh} + \cdots + a_{sh+r-1} X_{sh+r-1}. \end{aligned}$$

For $0 \leq i \leq s$, we define a set of tables $\mathsf{Tab}_X^{(i)}$ in the following manner: Define $\rho = (s+1)h$ and set $X_n = \cdots = X_{\rho-1} = 0$. Each table $\mathsf{Tab}_X^{(i)}$ stores 3^h points indexed by elements of $\{0, \pm 1\}^h$. For $(b_0, \ldots, b_{h-1}) \in \{0, \pm 1\}^h$, define

$$\mathsf{Tab}_X^{(i)}(b_0, \ldots, b_{h-1}) = b_0 X_{hi} + b_1 X_{hi+1} + \cdots + b_{h-1} X_{hi+h-1}. \quad (5)$$

Note that $X_n = \cdots = X_{\rho-1} = 0$ and hence $\mathsf{Tab}_X^{(s)}$ stores only 3^r points. The following algorithm can now be used to compute mX.

Algorithm 3.

Input : $m = \sum_{i=0}^{n-1} u_i \phi^i$, $n = s \times h + r$ and point X.

Output : mX.

1. Set $Y = \mathsf{Tab}_X^{(s)}[u_{sh,A}, u_{sh+1,A}, \ldots, u_{sh+r-1,A}, 0, \cdots, 0]$;
2. For $i = s-1$ down to 0 set $Y = Y + \mathsf{Tab}_X^{(i)}[u_{ih,A}, u_{ih+1,A}, \ldots, u_{(i+1)h-1,A}]$;
3. For $j = A-1$ down to 0
4. $\quad Y = 2Y$;
5. $\quad Y = Y + \mathsf{Tab}_X^{(s)}[u_{sh,j}, u_{sh+1,j}, \ldots, u_{sh+r-1,j}, 0, \cdots, 0]$;
6. $\quad$ For $i = s-1$ down to 0 set $Y = Y + \mathsf{Tab}_X^{(i)}[u_{ih,j}, u_{ih+1,j}, \ldots, u_{(i+1)h-1,j}]$;
7. End for;
8. return Y.

Proposition 3. *Algorithm 3 correctly computes mX using $(s+1)A$ additions and A doublings. For $0 \le i \le s-1$, table $\mathsf{Tab}_X^{(i)}$ stores 3^h points and $\mathsf{Tab}_X^{(s)}$ stores 3^r points. Thus the total number of points stored is $s \times 3^h + 3^r$.*

The storage requirement decreases from $3^n = 3^{sh+r}$ points to $s3^h + 3^r$ points. The trade-off is an increase in the number of additions. In the situation where the field F_{q^n} is represented using a normal basis, the storage requirement can be further reduced. This is based on the following observation.

Proposition 4. *For any $(b_0, \ldots, b_{h-1}) \in \{0, \pm 1\}^h$ and $i > 0$ we have,*

$$\mathsf{Tab}_X^{(i)}[b_0, \ldots, b_{h-1}] = \phi^{hi}(\mathsf{Tab}_X^{(0)}[b_0, \ldots, b_{h-1}]).$$

Proof: We compute

$$\begin{aligned}\mathsf{Tab}_X^{(i)}[b_0, \ldots, b_{h-1}] &= b_0 X_{hi} + b_1 X_{hi+1} + \cdots + b_{h-1} X_{hi+h-1} \\ &= b_0 \phi^{hi}(X) + b_1 \phi^{hi+1}(X) + \cdots + \phi^{hi+h-1}(X) \\ &= \phi^{hi}(b_0 X + b_1 \phi(X) + \cdots + b_{h-1}\phi^{h-1}(X)) \\ &= \phi^{hi}(\mathsf{Tab}_X^{(0)}[b_0, b_1, \ldots, b_{h-1}]).\end{aligned}$$

This completes the proof. $\square$

Since the field is represented using a normal basis, the map ϕ can be computed simply by a circular shift (see Section 2.3). Thus instead of storing the $(s+1)$ tables $\mathsf{Tab}_X^{(0)}, \ldots, \mathsf{Tab}_X^{(s)}$ we simply store the table $\mathsf{Tab}_X^{(0)}$ and for $i > 0$ we use Proposition 4 to compute any entry of $\mathsf{Tab}_X^{(i)}$ as and when required. Using this idea we obtain the following improvement.

Proposition 5. *Suppose the field F_{q^n} is represented using a normal basis. Then Algorithm 3 requires to store 3^h points. The numbers of additions and doublings remain the same as Proposition 3.*

4 General Table Look-Up Methods

In this section, we present our general table look-up algorithm. Let w ($1 \le w \le A+1$) be a positive integer which is the row window size and set $k = \lceil (A+1)/w \rceil$. We express all u_i occurring in the base-ϕ expansion of m in the base 2^w. In such an expansion we will use the elements of the set $\Omega_w = \{0, \pm 1, \pm 2, \cdots, \pm 2^{w-1}\}$ as digits. Note that since $\{0, \pm 1, \pm 2, \ldots, \pm(2^{w-1}-1), 2^{w-1}\}$ is a complete system of residues modulo 2^w, any integer m can be represented uniquely in base 2^w using these numbers as digits. The set Ω_w has one extra digit which ensures that the set is closed under negation. Thus, if $u = \sum_{i=0}^{t} a_i 2^{wi}$ then a representation of $-m$ over Ω_w can be obtained by simply negating all the a_i's. For $0 \le i \le n-1$, write

$$u_i = c_{i,0} + c_{i,1}2^w + \cdots + c_{i,k}2^{wk}, \tag{6}$$

where $c_{i,j} \in \Omega_w$. Then

$$\left.\begin{array}{ll} u_0X_0 & = (c_{0,0} + c_{0,1}2^w + \ldots + c_{0,k}2^{wk})X_0 \\ u_1X_1 & = (c_{1,0} + c_{1,1}2^w + \ldots + c_{1,k}2^{wk})X_1 \\ \vdots & \vdots\,\vdots \\ u_{n-1}X_{n-1} & = (c_{n-1,0} + c_{n-1,1}2^w + \ldots + c_{n-1,k}2^{wk})X_{n-1}. \end{array}\right\} \tag{7}$$

We have to compute $mX = u_0X_0 + \cdots + u_{n-1}X_{n-1}$. Let h ($1 \le h \le n$) be a small integer (which is the column window size) and write $n = s \times h + r$ where r is a unique integer in the set $\{1, \ldots, h\}$. We define $(s+1)$ tables $\mathsf{Tab}_X^{(0)}, \ldots, \mathsf{Tab}_X^{(s)}$, where each $\mathsf{Tab}_X^{(i)}$ stores $(2^w+1)^h$ points. The entries of $\mathsf{Tab}_X^{(i)}$ are indexed by elements of $\Omega_w^h = \underbrace{\Omega_w \times \cdots \times \Omega_w}_{h}$. For $(a_0, \ldots, a_{h-1}) \in \Omega_w^h$ we define

$$\mathsf{Tab}_X^{(i)}[a_0, \ldots, a_{h-1}] = a_0X_{ih} + a_1X_{ih+1} + \cdots + a_{h-1}X_{(i+1)h-1}. \tag{8}$$

Let $\kappa = (s+1)h$ and set $X_n = \cdots = X_{\kappa-1} = 0$. Hence $\mathsf{Tab}_X^{(s)}$ stores only $(2^w+1)^r$ points. With this set of tables at our disposal we can compute mX using Algorithm 4.

Algorithm 4.
Input : $m = \sum_{i=0}^{n-1} u_i\phi^i$ and point X .
Output : mX.

1. Set $Y = \mathsf{Tab}_X^{(s)}[c_{sh,k}, c_{sh+1,k}, \ldots, c_{n-1,k}, 0, \ldots, 0]$;
2. For $i = s-1$ down to 0
3. $\quad\quad Y = Y + \mathsf{Tab}_X^{(i)}[c_{ih,k}, c_{ih+1,k}, \ldots, c_{(i+1)h-1,k}]$;
4. For $j = k-1$ down to 0
5. $\quad\quad Y = 2^wY$;
6. $\quad\quad Y = Y + \mathsf{Tab}_X^{(s)}[c_{sh,j}, c_{sh+1,j}, \ldots, c_{n-1,j}, 0, \ldots, 0]$;
7. $\quad\quad$ For $i = s-1$ down to 0
8. $\quad\quad\quad\quad Y = Y + \mathsf{Tab}_X^{(i)}[c_{ih,j}, c_{ih+1,j}, \ldots, c_{(i+1)h-1,j}]$;
9. return Y.

Proposition 6. *Algorithm 4 correctly computes* mX *using* $(k-1)+ks$ *additions and* $(k-1)w$ *doublings. For* $0 \le i \le s-1$*, table* $\mathsf{Tab}_X^{(i)}$ *stores* $(2^w+1)^h$ *points and table* $\mathsf{Tab}_X^{(s)}$ *stores* $(2^w+1)^r$ *points. Thus a total of* $s(2^w+1)^h + (2^w+1)^r$ *points are required to be stored.*

As in Section 3.1, the storage requirement can be further reduced if the field F_{q^n} is represented using a normal basis. This is based on the following observation.

Proposition 7. *For any* $(a_0,\ldots,a_{h-1}) \in \Omega_w^h$*, and* $i > 0$*, we have*

$$\mathsf{Tab}_X^{(i)}[a_0,\ldots,a_{h-1}] = \phi^{hi}(\mathsf{Tab}_X^{(0)}[a_0,\ldots,a_{h-1}]).$$

Since the field is represented using a normal basis, the map ϕ is easy to compute online. Hence it is sufficient to store only $\mathsf{Tab}_X^{(0)}$ and compute the required entry of $\mathsf{Tab}_X^{(i)}$ as and when required. This gives us the following result.

Proposition 8. *If* F_{q^n} *is represented using a normal basis, then Algorithm 5 requires to store only* $(2^w+1)^h$ *points. The numbers of additions and doublings remain the same as in Proposition 6.*

5 Unknown Point

The algorithms described so far use one or more look-up tables. These tables are parametrized by a point X. If the point X is known in advance (as in signature generation for ElGamal algorithms), then the tables can be precomputed and stored. However, there are applications where the point is not known in advance (for example in variants of Diffie-Hellman key agreement protocols). In such a situation, the look-up tables have to be computed online. In this section, we describe algorithms for this task.

We start by describing an algorithm to compute the tables used in Algorithm 3. For this it is sufficient to describe an algorithm to compute $\mathsf{Tab}_X^{(0)}$. The Frobenius map can be used to compute the other tables from $\mathsf{Tab}_X^{(0)}$. Let X be a point and we wish to compute the table $\mathsf{Tab}_X^{(0)}$ having 3^h entries and indexed by the elements of the set $\{0,\pm1\}^h$. For any vector $\alpha \in \{0,\pm1\}^l$, we define $-\alpha$ to be the vector obtained from α by negating all the components of α. The following algorithm computes $\mathsf{Tab}_X^{(0)}$.

Algorithm 5.
input : X.
output : $\mathsf{Tab}_X^{(0)}$ used in Algorithm 3.

1. Compute $X_0,\ldots,X_{h-1}$.
2. $\mathsf{Tab}_X^{(0)}[0,0,\ldots,0] = 0$; $\mathsf{Tab}_X^{(0)}[1,0,\ldots,0] = X$; $\mathsf{Tab}_X^{(0)}[-1,0,\ldots,0] = -X$;
3. For $l = 1$ to $h-2$
4. For $\alpha \in \{0,\pm1\}^l$ set $\mathsf{Tab}_X^{(0)}[\alpha,1,\ldots,0] = X_{l+1} + \mathsf{Tab}_X[\alpha,0,\ldots,0]$;
5. For $\alpha \in \{0,\pm1\}^l$ set $\mathsf{Tab}_X^{(0)}[\alpha,-1,\ldots,0] = -\mathsf{Tab}_X[-\alpha,1,\ldots,0]$;
6. End.

Proposition 9. *Algorithm 5 correctly computes* $\mathsf{Tab}_X^{(0)}$ *used in Algorithm 3 using* $\frac{1}{2}(3^h - 3)$ *point additions and* $(h-1)$ *Frobenius map computations. The tables* $\mathsf{Tab}_X^{(1)}, \ldots, \mathsf{Tab}_X^{(s)}$ *used in Algorithm 3 can be computed from* $\mathsf{Tab}_X^{(0)}$ *using* $sh3^h$ *Frobenius map computations.*

Proof: First we prove the correctness. Let $\beta \in \{0, \pm 1\}$. If $\beta = (0, \ldots, 0)$, then clearly Algorithm 5 computes $\mathsf{Tab}_X^{(0)}[\beta] = 0$. So assume that $\beta \neq (0, \ldots, 0)$ and write $\beta = (\alpha, b, 0, \ldots, 0)$, where $b \neq 0$ and $\alpha \in \{0, \pm 1\}^l$ for some $l \geq 0$. We show that $\mathsf{Tab}_X^{(0)}[\beta]$ is computed correctly. If $b = 1$, we have by definition $\mathsf{Tab}_X^{(0)}[\beta] = \langle \alpha, (X_0, \ldots, X_l) \rangle + X_{l+1} = \mathsf{Tab}_X^{(0)}[\alpha, 0, \ldots, 0] + X_{l+1}$, where $\langle \rangle$ denotes the usual inner product. On the other hand, if $b = -1$, then

$$\begin{aligned} \mathsf{Tab}_X^{(0)}[\beta] &= \langle \alpha, (X_0, \ldots, X_l) \rangle - X_{l+1} \\ &= -(-\langle \alpha, (X_0, \ldots, X_l) \rangle + X_{l+1}) \\ &= -(\langle -\alpha, (X_0, \ldots, X_l) \rangle + X_{l+1}) \\ &= -(\mathsf{Tab}_X^{(0)}[-\alpha, 0, \ldots, 0] + X_{l+1}) \\ &= -\mathsf{Tab}_X^{(0)}[-\alpha, 1, \ldots, 0]. \end{aligned}$$

This completes the proof of correctness. Since $X_i = \phi^i(X)$, Step 1 of Algorithm 5 requires $h-1$ applications of the Frobenius map. The number of point additions is clearly $3 + 3^2 + \cdots + 3^{h-1} = \frac{1}{2}(3^h - 3)$.

Note that for any $\beta \in \{0, \pm 1\}^h$, and $i \geq 0$, we have $\mathsf{Tab}_X^{(i+1)}[\beta] = \phi^h(\mathsf{Tab}_X^{(i)}[\beta])$. To compute $\mathsf{Tab}_X^{(i+1)}$ from $\mathsf{Tab}_X^{(i)}$ we need $h3^h$ computations of the Frobenius map. Since s tables have to be computed, a total of $sh3^h$ computations of the Frobenius map is required. □

Note that if the field F_{q^n} is represented using normal basis, then for $i > 0$ the tables $\mathsf{Tab}_X^{(i)}$ need not be stored. Also the Frobenius map computation is essentially "for free".

Now we turn to the problem of computing the set of tables used in Algorithm 4. For $0 \leq i \leq h-1$ and $j \in \Omega_w$, we use the variable $Z_{i,j}$ to store the value of jX_i.

Algorithm 6
input : X;
output : $\mathsf{Tab}_X^{(0)}$ used in Algorithm 4.

1. For $i = 0$ to $h-1$, set $Z_{i,0} = 0$; $Z_{0,1} = X$;
2. For $j = 2$ to $2^w - 1$
3. $\quad Z_{0,j} = Z_{0,j-1} + X$; $Z_{0,-j} = -Z_{0,j}$;
4. End for;
5. For $i = 1$ to $h-1$
6. $\quad$ For $j = 1$ to $2^w - 1$
7. $\quad\quad Z_{i,j} = \phi(Z_{i-1,j})$; $Z_{i,-j} = -Z_{i,j}$;
8. $\quad$ End for;
9. End For;

10. For $j \in \Omega_w$, set $\mathsf{Tab}_X^{(0)}[j,0,\ldots,0] = Z_{0,j}$;
11. For $l = 1$ to $h-1$
12. For $\alpha \in \Omega_w^l$
13. For $j = 1$ to $2^w - 1$ set $\mathsf{Tab}_X^{(0)}[\alpha,j,0,\ldots,0] = Z_{l+1,j} + \mathsf{Tab}_X^{(0)}[\alpha,0,\ldots,0]$;
14. For $j = 1$ to $2^w - 1$ set $\mathsf{Tab}_X^{(0)}[\alpha,-j,0,\ldots,0] = -\mathsf{Tab}_X^{(0)}[-\alpha,j,0,\ldots,0]$;
15. End for;
16. End for;
17. End.

The following result whose proof is similar to that of Proposition 9 states the correctness and complexity of Algorithm 6.

Proposition 10. *Algorithm 6 correctly computes* $\mathsf{Tab}_X^{(0)}$ *used in Algorithm 4 using*

$$\left((2^w-2) + \frac{2^{2w}-1}{2^w}((2^w+1)^{h-1}-1)\right)$$

point additions and $(h-1)(2^w-1)$ *computations of the Frobenius map. Further, the tables* $\mathsf{Tab}_X^{(1)},\ldots,\mathsf{Tab}_X^{(s)}$ *used in Algorithm 4 can be computed using an additional* $sh(2^w+1)^h$ *computations of the Frobenius map.*

6 Results and Comparison

In this section, we present detailed results and also compare our algorithm with known scalar multiplication algorithms. At the outset, we would like to point out that the Frobenius map based method (and hence our algorithm) is really useful in the situation where the underlying field is a prime power field (rather than a prime field). Hence all our comparisons are to algorithms which work over prime power fields, in particular Optimial Extension Fields.

We recall the parameters of the algorithms (see Section 2.4): n is the field extension degree; h and w are respectively the column and row window sizes; $k = \lceil (A+1)/w \rceil$ and s, r are defined by the equation $n = s \times h + r$, where r is a unique integer from the set $\{1,\ldots,h\}$. Table 1 summarizes the results for scalar multiplication using Algorithm 4. Algorithm 3 is obtained from Algorithm 4 by putting $w = 1$. Further, Algorithm 2 is obtained from Algorithm 4 by putting $w = 1$ and $h = n$.

The first two columns of Table 1 gives the numbers of additions and doublings required. The third column gives the number of points required to be stored when normal basis is used and the fourth column gives the number of points required to be stored when standard (or polynomial) basis is used. Table 2 summarizes the result for Algorithm 6. The first half of Table 2 gives the numbers of additions and Frobenius map computations required to prepare the table $\mathsf{Tab}_X^{(0)}$ and the second half gives the additional number of Frobenius map computations required to prepare the tables $\mathsf{Tab}_X^{(1)},\ldots,\mathsf{Tab}_X^{(s)}$. Note that Algorithm 5 can be obtained from Algorithm 6 by setting $w = 1$. Let us denote the numbers of point additions

Table 1. Summary of Algorithm 4.

Additions	*Doublings*	*Normal basis*	*Standard basis*
$(k-1)+ks$	$(k-1)w$	$(2^w+1)^h$	$s(2^w+1)^h+(2^w+1)^r$

Table 2. Summary of Algorithm 6.

$\mathsf{Tab}_X^{(0)}$		$\mathsf{Tab}_X^{(1)},\ldots,\mathsf{Tab}_X^{(s)}$
Additions	*Frobenius map*	*Frobenius map*
$(2^w-2)+\frac{(2^{2^w}-1)}{2^w}\left((2^w+1)^{h-1}-1\right)$	$(h-1)(2^w-1)$	$sh(2^w+1)^h$

and point doublings by $\mathbf{A}$ and $\mathbf{D}$ respectively. From Table 1, $\mathbf{A}=(k-1)+ks$ and $\mathbf{D}=(k-1)w$. Using the fact that $k=\lceil (A+1)/w \rceil$ and $s \simeq \lfloor n/h \rfloor$ we have the $\mathbf{A}\simeq (n(A+1))/wh$ and $\mathbf{D}\simeq A$. The parameters w and h are respectively the row and column window sizes and hence wh is the size of the $w \times h$ submatrix window.

Define $t=n(A+1)$. Then the total number of bits required to represent the integer m in binary is $\simeq t$. The usual binary add-and-double algorithm requires t point doublings and on an average $(t/2)$ add point additions. A more efficient algorithm uses a non adjacent form (NAF) representation of m [23]. A window method using NAF and look-up table requires $t/(\omega+1)$ additions and t doublings while storing $2^{\omega-2}$ points, where ω is the window size (see [6]). The basic Frobenius map based algorithm (Algorithm 1) requires A doublings and $t/2$ additions. These results are summarized in Table 3 which clearly show the superiority of Algorithm 4 over the other algorithms. Algorithm 4 achieves the

Table 3. Comparison with other algorithms.

Algorithm	*additions*	*doublings*
Binary	max $(t-1)$; avg $(t/2)$	t
ω-NAF [6]	$t/(\omega+1)$	t
Algorithm 1	max $(t-1)$; avg $(t/2)$	A
Algorithm 4	$t/(wh)$	A

speed-up by using a look-up table. This look-up table can either be precomputed or can be computed online. Further, depending on the basis representation of the underlying field, the amount of storage space can vary.

In Table 4, we present results of storage and computational requirements under various conditions. The values in Table 4 clearly shows that the storage and computational requirements of the look-up tables can vary. For example, in the situation $n=4$, $h=2$ and $w=1$, it is sufficient to work with total storage space for 18 points (9 if normal basis is used). Only 3 point additions and 19

Frobenius map computations are required to compute the tables. The number of additions required for scalar multiplications is approximately $(t/wh) = t/2$ and the number of doublings is approximately A. This is better than the binary method. On the other hand, for $n = 12$, $h = 2$ and $w = 4$, using normal basis representation and storage for 289 points, the number of additions in the scalar multiplication can be brought down to $t/(wh) = t/8$. A total of 269 point additions are required to compute the tables in this situation. Thus Algorithm 4 provides a wide choice of trade-offs between storage space and efficiency of scalar multiplication.

Table 4. Storage and computation requirements of the look-up tables.

Parameters			*storage requirements*		*computational requirements*		
n	h	w	*normal basis*	*standard basis*	$\mathsf{Tab}_X^{(0)}$		$\mathsf{Tab}_X^{(i)}$ for $i > 0$
					Additions	*Frob*	*Frob*
4	4	1	81	81	39	3	0
		2	625	625	467	9	0
	2	1	9	18	3	1	18
		2	25	50	17	3	50
		3	81	162	69	7	162
8	4	1	81	162	39	3	324
		2	625	1250	467	9	2500
	2	1	9	36	3	1	54
		2	25	100	17	3	150
		3	81	324	69	7	486
12	4	1	81	243	39	3	648
		2	625	1875	467	9	5000
	2	1	9	54	3	1	90
		2	25	150	17	3	250
		3	81	486	69	7	810
		4	289	1734	269	15	2890

7 Conclusion and Further Research

We have described a new table look-up algorithm for performing Frobenius map based scalar multiplication for elliptic and hyperelliptic curve cryptosystems over prime power fields. The algorithm compares favourably with previous Frobenius map based algorithms and other scalar multiplication algorithms. Note that, we have not used NAF representation of the multiplier in our algorithm. Using NAF will further increase the efficiency of the proposed algorithm. Also, recently in [15], [19] more general techniques have been proposed for utilising the Frobenius map. It is an interesting work to see how the algorithm proposed in this work can be modified to suit the generalisation and how much of performance enhancement can be achieved. In conclusion, it can be said that our algorithm (or a version modified to suit the general scenario) is a serious contender for implementing scalar multiplication for elliptic and hyperelliptic curve cryptosystems.

References

1. K. Aoiki, F. Hoshino, T. Kobayashi A Cyclic Window Algorithm for ECC Defined over Extension Fields *ICICS 2001*, LNCS 2229, pp 62–73, Springer Verlag 2001.
2. D. V. Bailey and C. Paar. Efficient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography In *J. Cryptology 14*, pages 153–176, 2001.
3. D. G. Cantor. Computing in the Jacobian of a Hyperelliptic curve. In *Mathematics of Computation*,volume 48, pages 95–101, 1987.
4. Y. J. Choie and J. W. Lee. Speeding up the scalar multiplication in the Jacobian of hyperelliptic curves using Frobenius map. *Indocrypt 2002*, LNCS 2551, Springer Verlag 2002, pp 285–295.
5. M. Ciet, T. Lange, F. Sica and J.-J. Quisquater. Improved algorithms for efficient arithmetic on elliptic curves using fast endomorphisms. *Eurocrypt 2003.*
6. K. Fong, D. Hankerson, J. Lopez and A. Menezes. Field inversion and point halving revisited. *Preprint.*
7. R. P. Gallant, R. J. Lambert and S. A. Vanstone. Faster point multiplication on elliptic curves using efficient endomorphisms. *Crypto 2001*, LNCS 2139, pp. 190–200, 2001.
8. C. Gunther, T. Lange and A. Stein. Speeding Up the Arithmetic on Koblitz Curves of Genus Two, *Selected Areas in Cryptography, SAC 2001*, LNCS, pp. 106–117, 2001.
9. E. Knudsen. Elliptic scalar multiplication using point halving. *Proceedings of Asiacrypt 1999*, LNCS 1716, pp 135-149, 1999.
10. T. Kobayashi, H. Morita, K. Kobayashi and F. Hoshino. Fast elliptic curve algorithm combining Frobenius map and table reference to adapt to higher characteristic. *Eurocrypt 1999*, LNCS 1592, pp 176–189, 1999.
11. N. Koblitz. Hyperelliptic Cryptosystems. *Journal of Cryptology*, 1(3), pp 139–150, 1989.
12. N. Koblitz. *Algebraic Aspects of Cryptology.* Algorithm and Computation in Mathematics. Springer Verlag, 1998.
13. N. Koblitz. CM Curves with Good Cryptographic Properties. *Advances in Cryptology - Crypo'91*, LNCS 576, pp. 279–287,Springer Verlag 1992.
14. T. Lange. *Efficient Arithmetic on Hyperelliptic Koblitz Curve.* PhD thesis, University of Essen, 2001.
15. T. Park, E. Kim, K. Park, M. Lee. *A General Expansion Method Using Efficient Endomorphism.* In *Proceedings of ICISC, 2003*, LNCS, Springer-Verlag 2003.
16. R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, revised edition, 1994.
17. A. Menezes, Y. Wu and R. Zuccherato. An Elementary Introduction to Hyperelliptic Curve. Technical Report CORR 96–19, University of Waterloo (1996), Canada. Available at http://www.cacr.math.uwaterloo.ca
18. V. Müller. Fast multiplication on elliptic curves over small fields of characteristic two. *Journal of Cryptology*, 11(4):219–234, 1998.

19. T. Park, K. Park, M. Lee. *Efficient Scalar Multiplication in Hyperelliptic Curves using a new Frobenius Expansion.* In *Proceedings of ICISC, 2003*, LNCS, Springer-Verlag 2003.
20. R. Schroeppel. Elliptic curve point halving wins big. *Proceedings of 2nd Midwest Arithmetical Geometry in Cryptography Workshop.* Urbana, Illinois, November 2000.
21. F. Sica, M. Ciet and J.-J. Quisquater. Analysis of the Gallant-Lambert-Vanstone method based on efficient endomorphisms: elliptic and hyperelliptic curves. *Selected areas in cryptography.*, LNCS, 2002, to appear.
22. N. P. Smart. Elliptic curve cryptosystems over small fields of odd characteristic. *Journal of Cryptology*, 12(2):141-151, 1999.
23. J. Solinas. Efficient Arithmetic on Koblitz Curves. *Designs, Codes and Cryptography*, 19:195–249, 2000.
24. J. Solinas. An improved Algorithm on a Family of Elliptic Curves. In *Advances in Cryptology - Crpto'97*, LNCS 1294, pp.357–371, Springer-Verlag.