Endomorphisms for faster elliptic curve cryptography on a large class of curves

Steven D. Galbraith[⋆], Xibin Lin[⋆⋆], and Michael Scott[⋆ ⋆ ⋆]

[1] Mathematics Department,
Royal Holloway, University of London,
Egham, Surrey, TW20 0EX,
United Kingdom.
`steven.galbraith@rhul.ac.uk`

[2] School of Mathematics and Computational Science
Sun Yat-Sen University Guangzhou, 510275, P.R.China
`linxibin@mail2.sysu.edu.cn`

[3] School of Computing, Dublin City University.
Ballymun, Dublin 9, Ireland.
`mike@computing.dcu.ie`

Abstract. Efficiently computable homomorphisms allow elliptic curve point multiplication to be accelerated using the Gallant-Lambert-Vanstone (GLV) method. We extend results of Iijima, Matsuo, Chao and Tsujii which give such homomorphisms for a large class of elliptic curves by working over quadratic extensions and demonstrate that these results can be applied to the GLV method.
Our implementation runs in between 0.70 and 0.84 the time of the previous best methods for elliptic curve point multiplication on curves without small class number complex multiplication. Further speedups are possible when using more special curves.
Keywords: elliptic curves, point multiplication, GLV method, isogenies.

1 Introduction

Let E be an elliptic curve over a finite field $\mathbb{F}_q$ and let $P, Q \in E(\mathbb{F}_q)$ have order r. The fundamental operations in elliptic curve cryptography are point multiplication $[n]P$ and $[n]P + [m]Q$ where $n, m \in \mathbb{Z}$. There is a vast literature on efficient methods for computing $[n]P$ and $[n]P + [m]Q$ (a good reference is [3]). There is a significant difference between computing $[n]P$ for varying n and a fixed point P, and computing $[n]P$ where both n and P vary; this paper focusses on the latter case.

The Gallant-Lambert-Vanstone method [15] is an important tool for speeding up point multiplication. The basic idea is as follows. If the elliptic curve E has an efficiently computable endomorphism ψ (other than a standard multiplication by n map) such that $\psi(P) \in \langle P \rangle$ then one can replace the computation $[n]P$ by the multiexponentiation $[n_0]P + [n_1]\psi(P)$ where $|n_0|, |n_1| \approx \sqrt{r}$. The integers n_0 and n_1 are computed by solving a closest vector problem in a lattice, see [15] for details. In principle this computation requires only around 0.6 to 0.7 the time of the previous method (the precise details depend on the relative costs of doubling and addition, the window size being used, etc).

[⋆] This work supported by EPSRC grant EP/D069904/1.
[⋆⋆] This author thanks the Chinese Scholarship Council.
[⋆ ⋆ ⋆] This author acknowledges support from the Science Foundation Ireland under Grant No. 06/MI/006

Some examples allow higher degree decompositions such as $[n_0] + [n_1]\psi(P) + \cdots + [n_{m-1}]\psi^{m-1}(P)$ where $|n_i| \approx r^{1/m}$ which can give further speedups. We call the latter approach the m-dimensional GLV method.

Gallant, Lambert and Vanstone [15] only gave examples of suitable efficiently computable endomorphisms in two cases, namely subfield curves (i.e., groups $E(\mathbb{F}_{q^m})$ where E is defined over $\mathbb{F}_q$; these do not have prime or nearly prime order unless q is very small) and curves with special endomorphism structure (essentially, that the endomorphism ring has small class number). Hence, if one is using randomly chosen prime-order elliptic curves over finite fields for cryptography then the GLV method is not usually available. Indeed, in Section 7 of [30] one finds the claim "the GLV method is only effective for those exceptional elliptic curves that have complex multiplication by an order with small discriminant."

In fact, Iijima, Matsuo, Chao and Tsujii [21] constructed an efficiently computable homomorphism on elliptic curves $E(\mathbb{F}_{p^2})$ with $j(E) \in \mathbb{F}_p$ arising from the Frobenius map on a twist of E. Apparently they did not realise the application of their results to the GLV method. In this paper we give a generalisation of this approach and analyse it in the context of the GLV method. The techniques apply to all elliptic curves over $\mathbb{F}_{p^2}$ such that $j(E) \in \mathbb{F}_p$ and can be used with curves of prime order.

The curves considered in this paper are not completely general: the number of $\mathbb{F}_{q^2}$-isogeny classes of elliptic curves over $\mathbb{F}_{q^2}$ is approximately $2q^2$ whereas our construction gives only q isomorphism classes of curves. However, this is a major improvement over earlier papers on the GLV method which, in practice, were only applied to a finite number of $\mathbb{F}_q$-isomorphism classes for any given q. The results of this paper therefore overturn the claims of Section 7 of [30].

The basic idea is somewhat analogous to subfield curves: We take elliptic curves E with $j(E) \in \mathbb{F}_q$ and consider the group $E(\mathbb{F}_{q^m})$. However a crucial difference is that E is defined over $\mathbb{F}_{q^m}$, not $\mathbb{F}_q$. This means it is possible to obtain curves of prime order and so there is no need to restrict attention to q being small. Our method can be used with any prime power q and any elliptic curves E over $\mathbb{F}_q$ and always gives rise to a GLV method of dimension at least 2.

We give experimental results comparing our method for point multiplication $[n](x, y)$ with the best existing methods for this operation (indeed, we compare with optimised methods using only x-coordinate arithmetic due to Bernstein [4] and Gaudry-Thomé [17]). We find that the new method runs in between 0.70 and 0.84 the time of the previous best methods. The exact performance depends on the platform being used; our best result is for 8-bit processors. Our methods also can be applied in situations such as signature verification which are not covered by the timings in [4, 17] and we expect a significant speedup on previous methods in this case.

Note that our techniques can also be implemented on curves in Edwards form, and exploit their benefits. We also generalise the method to hyperelliptic curves.

The focus in this paper is on curves over prime fields, since in small characteristic one might prefer to use subfield curves and Frobenius expansions. However, Hankerson, Karabina and Menezes [20] have experimented with the method in characteristic 2 and they report that the new method runs in about 0.74 to 0.77 the time of the best standard method for general curves.

We now give an outline of the paper. First we describe the homomorphism and explain how it leads to a 2-dimensional GLV method. Section 3 gives a specific key generation algorithm which may be convenient for some applications. Section 4 shows how to get a 4-dimensional GLV method for $y^2 = x^3 + B$ over $\mathbb{F}_{p^2}$. Section 5 shows that our homomorphisms can also be used for curves written in Edwards form. Section 6 discusses the generalisation of our methods to higher genus curves. Section 7 discusses the algorithm we use for multiexponentiation. The proof of the pudding is the timings in Section 8. Section 9 discusses known security threats from using our construction and explains how to avoid them.

2 The homomorphism

We consider elliptic curves defined over any field $\mathbb{F}_q$ with point at infinity ∞. Recall that if E is an elliptic curve over $\mathbb{F}_q$ with $q+1-t$ points then one can compute the number of points $\#E(\mathbb{F}_{q^m})$ efficiently. For example, $\#E(\mathbb{F}_{q^2}) = q^2+1-(t^2-2q) = (q+1)^2-t^2$. As usual we define

$$E(\mathbb{F}_{q^m})[r] = \{P \in E(\mathbb{F}_{q^m}) : [r]P = \infty\}.$$

When we say that a curve or mapping is 'defined over $\mathbb{F}_{q^k}$' we mean that the coefficients of the polynomials are all in $\mathbb{F}_{q^k}$. The implicit assumption throughout the paper is that when we say an object is defined over a field $\mathbb{F}_{q^k}$ then it is not defined over any smaller field, unless explicitly mentioned.

The following result gives our main construction. Novices can replace the word 'separable isogeny' with 'isomorphism', set $d=1$ and replace $\hat{\phi}$ by ϕ^{-1} without any significant loss of functionality.

Theorem 1. *Let E be an elliptic curve defined over $\mathbb{F}_q$ such that $\#E(\mathbb{F}_q) = q+1-t$ and let $\phi : E \to E'$ be a separable isogeny of degree d defined over $\mathbb{F}_{q^k}$ where E' is an elliptic curve defined over $\mathbb{F}_{q^m}$ with $m \mid k$. Let $r \mid \#E'(\mathbb{F}_{q^m})$ be a prime such that $r > d$ and such that $r \| \#E'(\mathbb{F}_{q^k})$. Let π be the q-power Frobenius map on E and $\hat{\phi} : E' \to E$ be the dual isogeny of ϕ. Let*

$$\psi = \phi\pi\hat{\phi}.$$

Then

1. *$\psi \in End(E')$ (i.e., ψ is a group homomorphism).*
2. *For all $P \in E'(\mathbb{F}_{q^k})$ we have $\psi^k(P) - [d^k]P = \infty$ and $\psi^2(P) - [dt]\psi(P) + [d^2q]P = \infty$.*
3. *There is some $\lambda \in \mathbb{Z}$ such that $\lambda^k - d^k \equiv 0 \pmod{r}$ and $\lambda^2 - dt\lambda + d^2q \equiv 0 \pmod{r}$ such that $\psi(P) = [\lambda]P$ for all $P \in E'(\mathbb{F}_{q^m})[r]$.*

Proof. First note that $\hat{\phi}$ is an isogeny from E' to E and is defined over $\mathbb{F}_{q^k}$, that π is an isogeny from E to itself defined over $\mathbb{F}_q$, and that ϕ is an isogeny from E to E' defined over $\mathbb{F}_{q^k}$. Hence ψ is an isogeny of E' to itself, and is defined over $\mathbb{F}_{q^k}$ (or a subfield). Therefore, ψ is a group homomorphism.

Since $\phi\hat{\phi} = d$ on E' it follows that

$$\psi^2 = \phi\pi\hat{\phi}\phi\pi\hat{\phi} = \phi\pi d\pi\hat{\phi} = d\phi\pi^2\hat{\phi}$$

and, by induction, $\psi^k = d^{k-1}\phi\pi^k\hat{\phi}$. For $P \in E'(\mathbb{F}_{q^k})$ we have $\hat{\phi}(P) \in E(\mathbb{F}_{q^k})$ and so $\pi^k(\hat{\phi}(P)) = \hat{\phi}(P)$. Hence $\psi^k(P) = [d^k]P$.

Similarly, writing $Q = \hat{\phi}(P)$ we have $\pi^2(Q) - [t]\pi(Q) + [q]Q = \infty$ and so $[d]\phi(\pi^2 - [t]\pi + [q])\hat{\phi}(P) = \infty$. Using the previous algebra, this implies

$$(\psi^2 - [dt]\psi + [qd^2])P = \infty.$$

Finally, let $P \in E'(\mathbb{F}_{q^m})$ have order r. Since $\psi(P) \in E'(\mathbb{F}_{q^k})$ also has order r and $r \| \#E'(\mathbb{F}_{q^k})$ it follows that $\psi(P) = [\lambda]P$ for some $\lambda \in \mathbb{Z}$. Since ψ is a homomorphism, $\psi([a]P) = [a]\psi(P) = [\lambda]([a]P)$ for all $a \in \mathbb{Z}$. Since $\psi^k(P) - [d^k]P = [\lambda^k]P - [d^k]P = \infty$ it follows that $\lambda^k - d^k \equiv 0 \pmod{r}$. Similarly, $\lambda^2 - dt\lambda + d^2 q \equiv 0 \pmod{r}$. □

We stress that there is nothing unexpected in the above construction. Consider the case when ϕ is an isomorphism: Then $E' \cong E$ implies $\mathrm{End}(E') \cong \mathrm{End}(E)$. We know that $\mathrm{End}(E)$ contains the p-power Frobenius map and hence $\mathrm{End}(E')$ contains a corresponding endomorphism. The above Theorem simply writes down this endomorphism explicitly.

2.1 Special case of quadratic twists

We now specialise Theorem 1 to elliptic curves over $\mathbb{F}_p$ and the case $m = 2$, $p > 3$.

Corollary 1. *Let $p > 3$ be a prime and let E be an elliptic curve over $\mathbb{F}_p$ with $p + 1 - t$ points. Let E' over $\mathbb{F}_{p^2}$ be the quadratic twist of $E(\mathbb{F}_{p^2})$. Then $\#E'(\mathbb{F}_{p^2}) = (p-1)^2 + t^2$. Let $\phi : E \to E'$ be the twisting isomorphism defined over $\mathbb{F}_{p^4}$. Let $r \mid \#E'(\mathbb{F}_{p^2})$ be a prime such that $r > 2p$ Let $\psi = \phi\pi\phi^{-1}$. For $P \in E'(\mathbb{F}_{p^2})[r]$ we have $\psi^2(P) + P = \infty$.*

Proof. Let $E : y^2 = x^3 + Ax + B$ with $A, B \in \mathbb{F}_p$. We have $\#E(\mathbb{F}_{p^2}) = p^2 + 1 - (t^2 - 2p)$. Let $u \in \mathbb{F}_{p^2}$ be a non-square in $\mathbb{F}_{p^2}$, define $A' = u^2 A, B' = u^3 B$ and $E' : y^2 = x^3 + A'x + B'$. Then $\#E'(\mathbb{F}_{p^2}) = p^2 + 1 + (t^2 - 2p) = (p-1)^2 + t^2$. The isomorphism $\phi : E \to E'$ is defined by

$$\phi(x, y) = (ux, \sqrt{u}^3 y)$$

and is defined over $\mathbb{F}_{p^4}$.

If $r \mid \#E'(\mathbb{F}_{p^2})$ is prime such that $r > 2p$ then $r \nmid \#E(\mathbb{F}_{p^2}) = (p + 1 - t)(p + 1 + t)$ and so $r \| \#E'(\mathbb{F}_{p^4}) = \#E(\mathbb{F}_{p^2})\#E'(\mathbb{F}_{p^2})$. Hence we may apply Theorem 1 to get that ψ is a group homomorphism such that $\psi(P) = [\lambda]P$ such that $\lambda^4 - 1 \equiv 0 \pmod{r}$ for $P \in E'(\mathbb{F}_{p^2})[r]$. We now show that, in fact, $\lambda^2 + 1 \equiv 0 \pmod{r}$.

By definition, $\psi(x, y) = (ux^p/u^p, \sqrt{u}^3 y^p/\sqrt{u}^{3p})$ where $u \in \mathbb{F}_{p^2}$ (i.e., $u^{p^2} = u$) and $\sqrt{u} \notin \mathbb{F}_{p^2}$ (and so, $\sqrt{u}^{p^2} = -\sqrt{u}$). If $P = (x, y) \in E'(\mathbb{F}_{p^2})$ then $x^{p^2} = x, y^{p^2} = y$ and so

$$\begin{aligned}\psi^2(x, y) &= (ux^{p^2}/u^{p^2}, \sqrt{u}^3 y^{p^2}/\sqrt{u}^{3p^2}) \\ &= (x, (-1)^3 y) \\ &= -(x, y).\end{aligned}$$

This completes the proof. □

The above result applies to any elliptic curve over $\mathbb{F}_p$ (with $p > 3$) and shows that the 2-dimensional GLV method can be applied. Note that it is possible for $\#E'(\mathbb{F}_{p^2})$ to be prime, since E' is not defined over $\mathbb{F}_p$ (for further analysis see Nogami and Morikawa [26]). One feature of this construction is that, since p is now half the size compared with using elliptic curves over prime fields, point counting is much faster than usual (this was noted in [26]). Since we are dealing with elliptic curves over $\mathbb{F}_{p^2}$ where p is prime then Weil descent attacks are not a threat (see Section 9).

An exercise for the reader is to show that if E is an elliptic curve over $\mathbb{F}_p$ and if E' over $\mathbb{F}_p$ is the quadratic twist of E then the map ψ of our construction satisfies $\psi(P) = -P$ for all $P \in E'(\mathbb{F}_p)$. Our homomorphism is therefore useless for the GLV method in this case.

Corollary 2. *Let $p \equiv 5 \pmod 8$ be a prime. Let notation be as in Corollary 1. Then one may choose*

$$\psi(x, y) = (-x^p, iy^p)$$

where $i \in \mathbb{F}_p$ satisfies $i^2 = -1$.

Proof. We have $4 \| (p-1)$ and $2 \| (p+1)$. Since 2 is not a square in $\mathbb{F}_p$ one can define $\mathbb{F}_{p^2} = \mathbb{F}_p(u)$ where $u = \sqrt{2}$. Note that $u^p = -u$ and that $u^{p-1} \equiv 2^{(p-1)/2} \equiv -1 \pmod p$. It follows that $u^{(p^2-1)/2} = -1$ and so u is not a square in $\mathbb{F}_{p^2}$.

Since -1 is a square in $\mathbb{F}_p$ the equation $x^4 = 1$ has solutions $x = 1, -1, i, -i \in \mathbb{F}_p$. Let $w \in \mathbb{F}_{p^4}$ satisfy $w^2 = u$. Since $w \notin \mathbb{F}_{p^2}$ and $(w/w^p)^4 = 1$ we have $w^p = \pm iw$.

Finally, our homomorphism ψ is defined to be

$$\psi(x, y) = (ux^p/u^p, w^3y^p/w^{3p}) = (-x^p, \pm iy^p).$$

Renaming i if necessary gives the result. $\square$

Lemma 1. *Let notation be as in Corollary 1. Then $\psi(P) = [\lambda]P$ where $\lambda = t^{-1}(p-1) \pmod r$.*

Proof. The proof of Corollary 1 shows that $\psi(P) = [\lambda]P$ for some $\lambda \in \mathbb{Z}$. Since $\psi^2(P) = -P$ we have $\lambda^2 + 1 \equiv 0 \pmod r$. Similarly, $\psi^2(P) - [t]\psi(P) + [p]P = \infty$, so $\lambda^2 - t\lambda + p \equiv 0 \pmod r$. Subtracting the second equation from the first gives $t\lambda + (1-p) \equiv 0 \pmod r$.

Finally, we give some remarks about the lattice which arises in the GLV method when decomposing $[n]P$ as $[n_0]P + [n_1]\psi(P)$. Recall from [15] that we consider the lattice

$$L = \{(x, y) \in \mathbb{Z}^2 : x + y\lambda \equiv 0 \pmod r\}.$$

It is easy to prove that $\{(r, 0), (-\lambda, 1)\}$ is a basis for L; this shows that the determinant of L is r. The GLV method uses Babai's rounding method to solve the closest vector problem (CVP), and this method requires a reduced basis.

Lemma 2. *The vectors $\{(t, p-1), (1-p, t)\}$ are an orthogonal basis for a sublattice L' of L of determinant $\#E'(\mathbb{F}_{p^2})$. Given a point $(a, b) \in \mathbb{R}^2$ there exists a lattice point $(x, y) \in L'$ such that $\|(a, b) - (x, y)\| \leq (p+1)/\sqrt{2}$.*

Proof. In the proof of Lemma 1 we showed that $t\lambda + (1-p) \equiv 0 \pmod r$, which proves that $(1-p, t) \in L$. Multiplying by λ and using $\lambda^2 = -1$ gives $(t, p-1) \in L$. This basis has determinant $(p-1)^2 + t^2 = \#E'(\mathbb{F}_{p^2})$ so generates a sublattice $L' \subseteq L$ (if $\#E'(\mathbb{F}_{p^2}) = r$ then $L = L'$). It is easy to check that the basis vectors are orthogonal of length $\sqrt{\#E'(\mathbb{F}_{p^2})} \leq \sqrt{p^2 + 2p + 1} = p + 1$. Finally, simple geometry shows that the maximum distance from a lattice point is $\sqrt{\#E'(\mathbb{F}_{p^2})/2} \leq (p+1)/\sqrt{2}$.

Computing the coefficients n_0, n_1 for the GLV method is therefore particularly simple in this case. Further, one knows that $|n_0|, |n_1| \leq (p+1)/\sqrt{2}$. As always, an alternative to the lattice method which can be used in some cryptographic settings is to choose small coefficients $n_0, n_1 \in \mathbb{Z}$ directly rather than choosing a random $0 \leq n < r$ and then computing the corresponding (n_0, n_1).

2.2 Higher dimension decompositions

The GLV method can be generalised to m-dimensional decompositions $[n]P = [n_0]P + [n_1]\psi(P) + \cdots + [n_{m-1}]\psi^{m-1}(P)$ (for examples with $m = 4$ and $m = 8$ see [13]). Such a setting gives improved performance. As we have found 2-dimensional expansions using $E'(\mathbb{F}_{p^2})$ it is natural to try to get an m-dimensional decomposition using $E'(\mathbb{F}_{p^m})$.

In general, to obtain an m-dimensional decomposition it is required that ψ does not satisfy any polynomial equation on $E'(\mathbb{F}_{p^m})[r]$ of degree $< m$ with small integer coefficients. Note that ψ always satisfies a quadratic polynomial equation but that the coefficients are not necessarily small modulo r.

The following result gives a partial explanation of the behaviour of ψ on $E'(\mathbb{F}_{p^m})$.

Corollary 3. *Let $p > 3$ be a prime and let E be an elliptic curve over $\mathbb{F}_p$. Let E' over $\mathbb{F}_{p^m}$ be the quadratic twist of $E(\mathbb{F}_{p^m})$. Write $\phi : E \to E'$ for the twisting isomorphism defined over $\mathbb{F}_{p^{2m}}$. Let $r \mid \#E'(\mathbb{F}_{p^m})$ be a prime such that $r > 2p^{m/2}$ Let $\psi = \phi\pi\phi^{-1}$. For $P \in E'(\mathbb{F}_{p^m})[r]$ we have $\psi^m(P) + P = \infty$.*

Proof. As in Corollary 1, we have $r \| \#E'(\mathbb{F}_{p^{2m}}) = \#E'(\mathbb{F}_{p^m})\#E(\mathbb{F}_{p^m})$. Also, using the same method as the proof of Corollary 1 we have

$$\begin{aligned}\psi^m(x, y) &= (ux^{p^m}/u^{p^m}, \sqrt{u}^3 y^{p^m}/\sqrt{u}^{3p^m}) \\ &= -P.\end{aligned}$$

□

The problem is that the polynomial $x^m + 1$ is not usually irreducible, and it is possible that ψ satisfies a smaller degree polynomial. For example, in the case $m = 3$ one sees that $\#E'(\mathbb{F}_{p^3})$ cannot be prime as it is divisible by $N = \#E(\mathbb{F}_{p^2})/\#E(\mathbb{F}_p)$. If $r \mid \#E'(\mathbb{F}_{p^3})/N$ then $\psi^2(P) - \psi(P) + 1 = \infty$ for $P \in E'(\mathbb{F}_{p^3})[r]$. Hence one only gets a 2-dimensional decomposition in the case $m = 3$.

Indeed, the interesting case is when m is a power of 2, in which case $x^m + 1$ is irreducible and one can obtain an m-dimensional GLV decomposition. Indeed, Nogami and Morikawa [26] already proposed exactly this key generation method (choosing E over $\mathbb{F}_p$ and then using a quadratic twist over $\mathbb{F}_{p^{2^c}}$) as a method to generate curves of prime order. Note that [26] does not consider the GLV method.

Therefore, the next useful case is $m = 4$, giving a 4-dimensional GLV method. On the downside, this case is potentially vulnerable to Weil descent attacks (see Section 9) and so the prime p must be larger than we would ideally like.

The other way to get higher dimension decompositions is to have maps ϕ defined over larger fields than a quadratic extension. An example of this is given in Section 4.

3 Key generation

Let $p > 3$ be prime. We present a key generation algorithm for the quadratic twist construction. Our algorithm is designed so that the resulting curve $E' : y^2 = x^3 + A'x + B'$ over $\mathbb{F}_{p^2}$ has coefficient $A' = -3$, which is convenient for efficient implementation when using Jacobian coordinates (see Section 13.2 of [3]).

Algorithm 1 Key generation for quadratic twist construction

```
OUTPUT: $p, E', \psi, \lambda$
 1: Choose a prime $p = 5 \pmod 8$        ▷ $p$ can be special (e.g., a NIST prime; see Section 2.2.6 of [19])
 2: Set $u = \sqrt{2} \in \mathbb{F}_{p^2}$
 3: Set $A' = -3$ and $A = A'/2 \in \mathbb{F}_p$
 4: **repeat**
 5:     Choose random $B \in \mathbb{F}_p$ and let $E : y^2 = x^3 + Ax + B$
 6:     Compute $t = p + 1 - \#E(\mathbb{F}_p)$.
 7: **until** $(p-1)^2 + t^2 = hr$ where $r$ is prime and $h = 1$
 8: Set $B' = Bu^3 \in \mathbb{F}_{p^2}$ and $E' : y^2 = x^3 + A'x + B'$
 9: Set $\lambda = t^{-1}(p-1) \pmod r$
10: Compute $i \in \mathbb{F}_p$ so that $i^2 = -1$
11: Define $\psi(x,y) = (-x^p, iy^p)$.
12: **return** $p, (A', B'), \psi, \lambda$
```

We use Corollary 2, which gives a particularly simple map ψ. It should be clear that the algorithm can be used in more general cases. Our algorithm produces curves of prime order, but this can be relaxed by requiring only $h < H$ for some bound H in line 7.

As remarked earlier, key generation is fast compared with standard ECC, since the point counting for $\#E(\mathbb{F}_p)$ is over a field half the usual size (this is precisely the point of the paper [26]).

4 Using special curves

We have seen that one can obtain a 2-dimensional GLV method for any elliptic curve over $\mathbb{F}_p$. However, 2-dimensional GLV methods were already known for some special curves (i.e., those with a non-trivial automorphism or endomorphism of low degree). We now show how one can get higher-dimensional expansions using elliptic curves E over $\mathbb{F}_{p^2}$ with $\#\mathrm{Aut}(E) > 2$.

The two examples of interest are $E : y^2 = x^3 + B$ and $y^2 = x^3 + Ax$. We give the details in the former case. The latter is analogous.

Let $p \equiv 1 \pmod 6$ and let $B \in \mathbb{F}_p$. Define $E : y^2 = x^3 + B$. Choose $u \in \mathbb{F}_{p^{12}}$ such that $u^6 \in \mathbb{F}_{p^2}$ and define $E' : Y^2 = X^3 + u^6 B$ over $\mathbb{F}_{p^2}$. Repeat the construction (choosing p, B, u) until $\#E'(\mathbb{F}_{p^2})$ is prime (or nearly prime). Note that there are 6 possible group orders for $y^2 = x^3 + B'$ over $\mathbb{F}_{p^2}$ and three of them are never prime as they correspond to group orders of curves defined over $\mathbb{F}_p$.

The isomorphism $\phi : E \to E'$ is given by $\phi(x,y) = (u^2 x, u^3 y)$ and is defined over $\mathbb{F}_{p^{12}}$. The homomorphism $\psi = \phi\pi\phi^{-1}$, where π is the p-power Frobenius on E, is defined over $\mathbb{F}_{p^2}$ and satisfies the characteristic equation

$$\psi^4 - \psi^2 + 1 = 0.$$

Hence one obtains a 4-dimensional GLV method for these curves. This leads, once again, to a major speedup of these curves compared with previous techniques.

Note that $-\psi^2$ satisfies the characteristic equation $x^2 + x + 1$ and so acts as the standard automorphism $(x,y) \mapsto (\zeta_3 x, y)$ on E.

5 Using Edwards curves

In this section we explain how to write our homomorphism in terms of the Edwards equation for an elliptic curve, under the assumption that our original curve E has a point of order 4 and a unique point of order 2 (these conditions imply that E can be written in Edwards form). This means that the multiexponentiation can be performed using the Edwards formulae for elliptic curve additions and doublings. We closely follow Section 2 of [7].

Let E be an elliptic curve $E/\mathbb{F}_{p^2} : y^2 = x^3 + Ax + B$ with a point $R = (r_1, s_1)$ defined over $\mathbb{F}_{p^2}$ of order 4 and a unique point $S = [2]R = (r_2, 0)$ of order 2. Suppose we have an efficiently computable homomorphism $\psi : E(\mathbb{F}_{p^2}) \to E(\mathbb{F}_{p^2})$ of the form $\psi : (x, y) = (c_1 x^p, c_2 y^p)$, where $c_1, c_2 \in \mathbb{F}_{p^2}$ are constants. We assume that $\psi(P) = [\lambda]P$ for $P \in E(\mathbb{F}_{p^2})[r]$ for some $\lambda \in \mathbb{Z}$. We will transform E to an Edwards elliptic curve E_e with an efficiently computable homomorphism ψ_e on it, such that $\psi_e(P) = [\lambda]P$ for $P \in E_e(\mathbb{F}_{p^2})[r]$.

We first move S to $(0, 0)$ using the standard transformation $\chi_1(x, y) = (x - r_2, y)$ which maps to $E_1 : Y^2 = X^3 + a_2X^2 + a_4X$, where $a_2 = 3r_2$ and $a_4 = 3r_2^2 + A$ are defined over $\mathbb{F}_{p^2}$.

As in [7], let $d = 1 - \frac{4r_1^3}{s_1^2} \in \mathbb{F}_{p^2}$ and consider the elliptic curve $E_e : \bar{x}^2 + \bar{y}^2 = 1 + d\bar{x}^2\bar{y}^2$ in Edwards form. We now present the explicit birational map from $E_1 \to E_e$ given in [7].

Let $t_1 = \sqrt{\frac{r_1}{1-d}} = \pm\frac{s_1}{2r_1}$. Define $E_2 : (\frac{r_1}{1-d})y^2 = x^3 + a_2x^2 + a_4x$. Then there is an isomorphism χ_2 defined over $\mathbb{F}_{p^2}$ from $E_1 \to E_2$ by $\chi_2(x, y) = (x, y/t_1)$. As explained in [7], we know that $a_4 = r_1^2$ and $a_2 = 2r_1(1 + d)/(1 - d)$. Let $E_3 : (\frac{1}{1-d})y^2 = x^3 + 2((1 + d)/(1 - d))x^2 + x$. The isomorphism χ_3 from E_2 to E_3 over $\mathbb{F}_{p^2}$ is given by $\chi_3(x, y) = (x/r_1, y/r_1)$. Finally, E_3 is birationally equivalent over $\mathbb{F}_{p^2}$ to E_e by the birational map $\chi_4(x, y) = (2x/y, (x - 1)/(x + 1))$.

To summarize the above, there is a birational map ρ defined over $\mathbb{F}_{p^2}$ from E to E_e given by

$$\rho(x, y) = \left(\frac{2t_1(x - r_2)}{y}, \frac{x - r_2 - r_1}{x - r_2 + r_1}\right) = (\bar{x}, \bar{y}).$$

The birational map ρ^{-1} from E_e to E is given by $\rho^{-1}(\bar{x}, \bar{y}) = (\frac{(r_2-r_1)\bar{y}-(r_1+r_2)}{\bar{y}-1}, \frac{-2r_1t_1(\bar{y}+1)}{\bar{x}(\bar{y}-1)})$.

We may now define the homomorphism $\psi_e = \rho\psi\rho^{-1}$. We will compute an explicit form for ψ_e. Let $a = r_1 + r_2$ and $b = r_2 - r_1$. Then

$$\begin{aligned}
\psi_e(\bar{x}, \bar{y}) &= \rho\psi\rho^{-1}(\bar{x}, \bar{y}) \\
&= \rho\left(\frac{c_1(b\bar{y} - a)^p}{(\bar{y} - 1)^p}, \frac{c_2(-2r_1t_1(\bar{y} + 1))^p}{(\bar{x}(\bar{y} - 1))^p}\right) \\
&= \left(\frac{2t_1(c_1(\frac{b\bar{y}-a}{\bar{y}-1})^p - r_2)}{c_2(\frac{-2r_1t_1(\bar{y}+1)}{\bar{x}(\bar{y}-1)})^p}, \frac{c_1(\frac{b\bar{y}-a}{\bar{y}-1})^p - a}{c_1(\frac{b\bar{y}-a}{\bar{y}-1})^p - b}\right) \\
&= \left(\frac{\bar{x}^p(m_1 - m_2\bar{y}^p)}{m_3(\bar{y}^p + 1)}, \frac{m_4\bar{y}^p - m_5}{m_6\bar{y}^p - m_7}\right)
\end{aligned}$$

where $m_1 = c_1a^p - r_2$, $m_2 = c_1b^p - r_2$, $m_3 = c_2r_1^pt_1^{p-1}$, $m_4 = c_1b^p - a$, $m_5 = c_1a^p - a$, $m_6 = c_1b^p - b$, and $m_7 = c_1a^p - b$ are constants in $\mathbb{F}_{p^2}$ which may be precomputed.

It follows that ψ_e can be computed naively using two Frobenius computations, 5 multiplications and two inversions. We can also use Montgomery's trick to replace two inversions with one inversion and one multiplication. So the homomorphism ψ_e is certainly efficiently computable.

6 Hyperelliptic curves

Afficionados will have noticed that Theorem 1 holds (with minor modifications to the second part of property (2)) for arbitrary abelian varieties (this has been noted by Kozaki, Matsuo and Shimbara [22], but they do not use it for the GLV method). We now present an analogue of Corollary 1 for hyperelliptic curves.

Let $C : y^2 = x^{2g+1} + f_{2g}x^{2g} + \cdots + f_1x + f_0$ be a genus g curve over $\mathbb{F}_q$ with a single point at infinity. Consider the Jacobian of C over $\mathbb{F}_{q^m}$ and take a quadratic twist $C' : y^2 = x^{2g+1} + uf_{2g}x^{2g} + \cdots + u^{2g}x + u^{2g+1}f_0$ where $u \in \mathbb{F}_{q^m}$ is a non-square. The isomorphism $\psi : C \to C'$ is given by

$$\phi(x, y) = (ux, \sqrt{u}^{2g+1}y)$$

This map induces an isomorphism $\phi : \mathrm{Jac}(C) \to \mathrm{Jac}(C')$ over $\mathbb{F}_{q^{2m}}$ which can be explicitly calculated on the Mumford representation (see [22]).

Our construction gives the homomorphism $\psi = \phi\pi\phi^{-1}$ satisfying $\psi^m(D) + D = 0$ for $D \in \mathrm{Jac}(C')(\mathbb{F}_{q^m})$ and therefore, when m is a power of 2, one obtains an m-dimensional GLV method.

In this case, the speedup for key generation is crucial: counting the number of points on random Jacobians of cryptographic size in large characteristic is currently impractical, however our new approach is certainly feasible in practice.

On the other hand, Weil descent attacks are much more successful in higher genus. Indeed, as discussed in Section 9, even the case $m = 2$ is potentially vulnerable to Weil descent attacks. Hence one needs to increase the size of q to attain the required security level. A careful implementation is required to determine the advantages (if any) in this case.

7 Multiexponentiation

The point of the GLV method is to replace a large point multiplication $[n]P$ by a multiexponentiation $[n_0]P + [n_1]\psi(P)$. There are numerous algorithms for multiexponentiation and much has been written on the topic. One approach is to use 'interleaving'; this was first proposed in [15] and is also described in [19]. These methods seem to be better in practice than using the joint sparse form of Solinas [32]. We give further analysis of multiexponentiation methods in an appendix.

We propose a combination of interleaving and non-adjacent form (NAF) expansions which we believe, for the parameters and implementation platforms considered in this paper, gives the best efficiency. It requires relatively little precomputation and is very simple to implement. First, we recall the definition of the NAF.

Definition 1 (Definition 3.28 [19]). *A non-adjacent form (NAF) of a positive integer k is an expression $k = \sum_{i=0}^{t} k_i 2^i$ where $k_i \in \{0, \pm 1\}$, $k_t \neq 0$, and no two consecutive digits k_i are nonzero. The length of the NAF is $t + 1$.*

A very efficient method (as it only uses a few word operations) to compute the NAF of an integer n is to compute $3n$ (using standard integer multiplication), then form the signed expansion $(3n) - n$ and discard the least significant bit. It is well-known that the average Hamming weight of a NAF of length t is $t/3$. See Section 9.1.4 of [3] for details.

Our method uses sliding windows over NAF expansions (this idea is also mentioned on page 101 of [19]). Let $\omega \geq 2$ be the window size and let W be the largest odd integer representing a NAF of length ω with non-zero first and last coefficients (e.g., when $\omega = 4$ then $W = (1001)_2 = 9$). We will then require a total of $(W + 1)/2 = (2^\omega - (-1)^\omega)/3$ locations to store the precalculated points $\{P, [3]P, \ldots, [W]P\}$ [19]. Given a NAF $(k_t, k_{t-1}, \ldots, k_0)$ one can compute from right to left a sequence $(k'_t, \ldots, k'_0)$ such that $\sum_{i=0}^{t} k'_i 2^i = \sum_{i=0}^{t} k_i 2^i$, $k'_i \in \{0, \pm 1, \pm 3, \ldots, \pm W\}$ and among any ω consecutive coefficients k'_i at most one is non-zero. In practice one prefers to compute from left to right a sequence $(k'_t, \ldots, k'_0)$ such that $\sum_{i=0}^{t} k'_i 2^i = \sum_{i=0}^{t} k_i 2^i$ and $k'_i \in \{0, \pm1, \pm3, \ldots, \pm W\}$; this sequence can be different to the one above, but it has the same proportion of non-zero coefficients. We call the coefficients $(k'_t, \ldots, k'_0)$ the sliding windows of length ω over the NAF.

Our proposal to compute $[n_0]P + [n_1]\psi(P)$ is as follows. We first compute the NAF representation of n_0 and n_1, we then perform an interleaved multiexponentiation, where the computations $[n_0]P$ and $[n_1]\psi(P)$ are performed using sliding windows of width ω over the NAF expansion. We call the algorithm "interleaving ω-sliding window NAF" (I-ω-SW-NAF). The details are given in Algorithm 2 (in greater generality, since the method is also useful for signature verification without using the GLV idea). Steps 1 and 2 of the algorithm are performed using methods of Dahmen, Okeya, and Schepers [12].

It may seem counterintuitive to window the NAF, rather than compute a ω-NAF, especially since the ω-NAF is proven to have the minimal number of non-zero coefficients among such expansions. The subtlety is that the standard definition for an ω-NAF is to have coefficients $n_{i,j}$ such that $|n_{i,j}| < 2^{\omega-1}$ whereas our method has coefficients $|n_{i,j}| < 2^{\omega}$, but not every coefficient can arise. Hence our method is like an intermediate between the ω-NAF and $(\omega+1)$-NAF. This approach seems to offer a good balance between precomputation cost, cost of computing the NAF, and time for the multiexponentiation.

Algorithm 2 Interleaving ω-sliding window NAF

```
INPUT: $n_0$, $n_1$, $P, Q$ and widths $\omega_0, \omega_1$
OUTPUT: $[n_0]P + [n_1]Q$
 1: Compute $[i]P$ for $i \in \{1, 3, \cdots, W_0\}$, where $W_0$ is largest positive odd integer in a length $w_0$ NAF string.
 2: Compute $[i]Q$ for $i \in \{1, 3, \cdots, W_1\}$,where $W_1$ is largest positive odd integer in a length $w_1$ NAF string.
 3: Compute the NAF $\sum_{i=0}^{t_0} n_{0,i} 2^i$ of $n_0$ (where $n_{0,i} \in \{-1, 0, 1\}$)
 4: Compute the NAF $\sum_{i=0}^{t_1} n_{1,i} 2^i$ of $n_1$ (where $n_{1,i} \in \{-1, 0, 1\}$)
 5: Let $t = \max\{t_0, t_1\}$. Set $n_{0,i} = 0$ for $t_0 < i \le t$ and $n_{1,i} = 0$ for $t_1 < i \le t$
 6: $R = \infty$
 7: for $i$ from $t$ downto 0 do
 8:     $R \leftarrow [2]R$.
 9:     Compute $i$-th sliding window coefficient $n'_{0,i} \in \{0, \pm1, \pm3, \ldots, \pm W_0\}$ the NAF of $n_0$
10:     if $n'_{0,i} \neq 0$ then
11:         $R = R + [n'_{0,i}]P$        ▷ When $n'_{0,i} < 0$, $[n'_{0,i}]P$ can be easily obtained from $[-n'_{0,i}]P$
12:     end if
13:     Compute $i$-th sliding window coefficient $n'_{1,i} \in \{0, \pm1, \pm3, \ldots, \pm W_1\}$ over the NAF of $n_1$
14:     if $n'_{1,i} \neq 0$ then
15:         $R = R + [n'_{1,i}]Q$        ▷ When $n'_{1,i} < 0$, $[n'_{1,i}]Q$ can be easily obtained from $[-n'_{1,i}]Q$
16:     end if
17: end for
18: return R
```

We give an estimate of the time for multiexponentiation when using elliptic curves over $\mathbb{F}_{p^m}$. Denote by $\mathsf{M}, \mathsf{S}, \mathsf{I}$ the costs of a multiplication, squaring and inversion in $\mathbb{F}_{p^m}$ respectively. One can check that, when $m = 2$, $\mathsf{M} = 3\mathsf{m}$ and $\mathsf{S} = 2\mathsf{m}$ where m is the cost of an $\mathbb{F}_p$ multiplication. Note that $\mathbb{F}_p$ squarings do not arise in these calculations.

When using Jacobian projective coordinates one prefers to use mixed additions in the main loop as they are faster. However this requires that any precomputed values must be "normalized", that is converted to affine form, before entering the loop. This conversion, if done naively for each precomputed point, would require an expensive field inversion, so care must be taken to minimize the impact of (or otherwise seek to avoid) this normalization step. See [9] for a detailed discussion. For example when using the JSF method to calculate $[a]P + [b]Q$ with P and Q in affine form, it is required to precompute $P + Q$ and $P - Q$. It is an easy exercise to show that these two points can in fact be calculated jointly using a single base field inversion, and the additional cost of $4\mathsf{M} + 2\mathsf{S}$, which is $4(3\mathsf{m}) + 2(2\mathsf{m}) = 16\mathsf{m}$.

When using window methods, increasing the window size reduces the number of point additions/subtractions in the main loop, but it also increases the amount of online precomputation required. So there is an optimal window size, which increases slowly with the length in bits of the multiplier.

Using standard Jacobian coordinates, a point doubling costs $4\mathsf{M} + 4\mathsf{S}$, which becomes $4(3\mathsf{m}) + 4(2\mathsf{m}) = 20\mathsf{m}$ when working in $E(\mathbb{F}_{p^2})$, and a mixed point addition costs $8\mathsf{M}+3\mathsf{S} = 8(3\mathsf{m})+3(2\mathsf{m}) = 30\mathsf{m}$.

We use the precomputation strategy of Dahmen, Okeya and Schepers (DOS) [12], as recommended in [9], which again requires only a single inversion. For a 4-bit sliding window method, a total of 5 points $\{P, [3]P, [5]P, [7]P, [9]P\}$ need to be precalculated and stored. From [12] we find that the precomputation cost is then $39\mathsf{M} + 20\mathsf{S} = 39(3\mathsf{m}) + 20(2\mathsf{m}) = 157\mathsf{m}$. From these precomputed values, the values $\{\psi(P), [3]\psi(P), [5]\psi(P), [7]\psi(P), [9]\psi(P)\}$ can be quickly determined; when working in $E(\mathbb{F}_{p^2})$ the further cost is at most $30\mathsf{m}$ for the five applications of ψ. The main loop now requires on average only one point doubling plus 0.381 mixed additions per bit, and the total cost can be estimated as $4147\mathsf{m}$. The constant $c = 0.381$ is derived from the formula given in [19], immediately after algorithm 3.38. For a window size of ω the average length of a run of zeros between windows (which we will "slide" over) is given as

$$v(\omega) = \frac{4}{3} - \frac{(-1)^w}{3.2^{\omega-2}}$$

Then $c = 2/(\omega + v(\omega))$, as we are interleaving two sliding windows, one for P, and one for $\psi(P)$.

Note that for a 5-bit sliding window the estimated cost is $4157\mathsf{m}$, just marginally slower. There is also to be considered the cost of two inversions, one associated with the precomputation, and another at the end of the calculation to finally return a result in affine coordinates.

8 Experimental results

We now give some timing comparisons for the computation of $[n]P$ (and also signature verification) on elliptic curves at the 128-bit security level. Our timings are for the case of quadratic twists as presented in Section 2.1.

8.1 The example curve

It is natural to use the Mersenne prime $p = 2^{127} - 1$, which is also used in Bernstein's `surface1271` genus 2 implementation [6][4]. This prime supports a very fast modular reduction algorithm. Note

[4] Note that the Pollard rho algorithm using equivalence classes in this case requires approximately 2^{125} group operations, the same as for Bernstein's Curve25519 or Surface1271. Whether this is precisely the same security level as AES-128 is unclear, but since Curve25519 and Surface1271 have been used for benchmarking we feel our choice is justified.

that since $p \neq 5 \bmod 8$ the previously described key generation process is not applicable here. However it can easily be modified to handle this case as well, although the homomorphism requires more multiplications to compute.

Let

$$E : y^2 = x^3 - 3x + 44$$

be defined over the field $\mathbb{F}_p$. Then $\#E(\mathbb{F}_p) = p + 1 - t$ where $t = 3604275729619761575$ (or, in hex, `3204F5AE088C39A7`). By Corollary 1 the quadratic twist E' over $\mathbb{F}_{p^2}$ of $E(\mathbb{F}_{p^2})$ has $\#E'(\mathbb{F}_{p^2}) = (p-1)^2 + t^2$, which is a prime we call r. The curve was quickly found using a modified version of Schoof's algorithm.

Since $p \equiv 7 \bmod 8$, we can choose to represent $\mathbb{F}_{p^2}$ as $\mathbb{F}_p(i)$ where $i^2 = -1$. So now choose $u = 2 + i$ which is the non-square in $\mathbb{F}_{p^2}$. Note also that $p \equiv 2 \bmod 5$. Our homomorphism is this case simplifies to

$$\psi(x, y) = (\omega_x \bar{x}, \omega_y \bar{y})$$

where $\bar{x}$ denotes the conjugate of x, and $\omega_x = u/u^p, \omega_y = \sqrt{u^3/u^{3p}}$ as in the proof of Corollary 1. By Lemma 1 we have $\psi(P) = [\lambda]P$ where $\lambda = t^{-1}(p-1) \pmod{r}$.

8.2 Theoretical Comparison

For comparison purposes we initially consider an elliptic curve $E(\mathbb{F}_p)$ defined with respect to the 256-bit pseudo-Mersenne modulus $p = 2^{256} - 189$, which provides approximately the same level of security as the curve in the previous subsection. We will compare operation counts for a full point multiplication. (An earlier version of this paper used a NIST standard curve for comparison purposes, but pseudo-mersenne moduli allow for a much faster modular reduction, and thus makes for a fairer comparison.) Our implementation uses Jacobian coordinates, as specified in the IEEE-1363 standard. Whereas improved parameterisations and formulae are available, we point out that these would benefit both implementations more or less equally. (Having said that, when the point doubling and addition formulae are implemented over $\mathbb{F}_{p^2}$ we note that field squarings take 2/3 the time of field multiplications, and that field inversions are not quite as bad as they would be over $\mathbb{F}_p$, for a double-length p, and hence implementations based on affine coordinates may be of interest in very space constrained environments.)

Recall that the calculation of interest is $[k_0]P + [k_1]\psi(P)$, where k_0 and k_1 arise as a result of the decomposition of a larger random $k < r$. We assume that P is initially presented in affine coordinates. Note that k_0 and/or k_1 might be negative – but this is not a problem as $[-a]P = [a](-P)$, and point negation on an elliptic curve is essentially free.

As discussed in Section 7, we use an interleaving algorithm for multiexponentiation using NAFs windowed with sliding windows of length 4. In practise "fractional windows" are probably to be preferred [9], [25], as they allow for finer control over the optimal window size, but they are more difficult to analyse, and in practise any improvement achieved by using fractional windows was found to be negligible.

The total cost of the exponentiation is expected to be $4147\mathsf{m}$ where m is the cost of a 128-bit modular multiplication.

For the $E(\mathbb{F}_p)$ (256-bit p) curve we find that a window size of 5 is slightly better, and we use a standard sliding window NAF method and random 254-bit multipliers. The expected cost will be approximately $(4\mathcal{M} + 4\mathcal{S}) + 0.157(8\mathcal{M} + 3\mathcal{S})$ per bit where $\mathcal{M}$ is the cost of a 256-bit $\mathbb{F}_p$ multiplication and $\mathcal{S}$ the cost of a 256-bit $\mathbb{F}_p$ squaring. For simplicity we will assume that

$\mathcal{M} = \mathcal{S}$. The precomputation (again using the DOS method) costs $143\mathcal{M}$ and so the total cost for a point multiplication can be estimated as $2614\mathcal{M}$. Again two inversions are required, one for the precomputation, one to render the final result in affine coordinates. As the number of inversions is small, and the same in all cases, we neglect its contribution.

It is useful to also compare with the operation counts that arise when implementing straightforward point multiplication on $E(\mathbb{F}_{p^2})$ *without* using the new homomorphism, so we will include these as well. In this case we use a standard sliding windows algorithm, again with a window size of 5.

In our implementations we averaged the cost over 10^5 point multiplications, and the results are presented in Table 1. Here SSW indicates use of standard sliding windows, GLV indicates the use of our homomorphism with the GLV decomposition, JSF means that the Joint Sparse Form method was used, and INT refers to the use of the interleaving algorithm of Section 7.

Table 1. Point multiplication operation counts

	Method	$\mathbb{F}_p$ muls	$\mathbb{F}_p$ adds/subs
$E(\mathbb{F}_p)$, 256-bit p	SSW	2600	3775
$E(\mathbb{F}_{p^2})$, 127-bit p	SSW	6641	16997
$E(\mathbb{F}_{p^2})$, 127-bit p	GLV+JSF	4423	10785
$E(\mathbb{F}_{p^2})$, 127-bit p	GLV+INT	4109	10112

The results in Table 1 support our rough analysis above. However we also include in this table the often neglected costs of field additions and subtractions. Note that when implementing $\mathbb{F}_{p^2}$ arithmetic, each multiplication using Karatsuba also requires five $\mathbb{F}_p$ additions/subtractions, so the number of these operations increases substantially.

Clearly the superiority (or otherwise) of the new method depends on the ratio $\mathsf{m}/\mathcal{M}$, in other words, the efficiency with which 128-bit and 256-bit field multiplications and additions/subtractions can be implemented on particular platforms.

To give a more accurate picture we have implemented both methods on two widely differing platforms, a 1.66GHz 64-bit Intel Core 2, and on an 8-bit 4MHz Atmel Atmega1281 chip (which is a popular choice for Wireless Sensor Network nodes). We present the results in the following two subsections.

8.3 8-bit processor implementation

Our first implementation is on a small 4MHz 8-bit Atmega1281 processor. Here the base field multiplication times will dominate, so this function was written in optimal loop-unrolled assembly language. We use the MIRACL C library [28], which includes tools for the automatic generation of such code (and which holds the current speed record in its class for this particular processor[29]), and we use the cycle accurate AVR Studio tool to measure the time for a single variable point multiplication.

As can be seem from these timings in table 2, our best method for ECC point multiplication takes about 0.70 of the time required for the 256 bit $E(\mathbb{F}_p)$ curve.

Observe that simply switching to an $E(\mathbb{F}_{p^2})$ curve at the same security level does not by itself give any improvement, in fact it is somewhat slower. Digging deeper we quickly find the reason -

Table 2. Point multiplication timings – 8-bit processor

Atmel Atmega1281 processor	Method	Time (s)
$E(\mathbb{F}_p)$, (256-bit p)	SSW	5.49
$E(\mathbb{F}_{p^2})$ (127-bit p)	SSW	6.20
$E(\mathbb{F}_{p^2})$, (127-bit p)	GLV+JSF	4.21
$E(\mathbb{F}_{p^2})$, (127-bit p)	GLV+INT	3.87

a field multiplication takes 1995 μs over $\mathbb{F}_p$ (256-bit p), as against 2327 μs over $\mathbb{F}_{p^2}$ (127-bit p), although for a field squaring the situation is reversed, taking 1616 μs over $\mathbb{F}_p$ as against only 1529 μs over $\mathbb{F}_{p^2}$. Field addition and subtraction favours the $\mathbb{F}_p$ case (124 μs versus 174 μs).

However using the new homomorphism and applying the GLV method, our new method is still clearly superior.

Note that for this processor it is probably more appropriate in practise to use the JSF method for point multiplication, as it is much better suited to a small constrained enviroment, with limited space for online precomputation.

8.4 64-bit processor implementation

It has been observed by Avanzi [2], that software implementations over smaller prime fields, where field elements can be stored in just a few CPU registers (as will be the case here), suffer disproportionally when implemented using general purpose multi-precision libraries. This effect would work against us here, as we are using the general purpose MIRACL library [28]. Special purpose libraries like the `mpFq` library [17] which generate field-specific code, and implementations which work hard to squeeze out overheads, such as Bernstein's implementations [6] are always going to be faster.

In the context of a 64-bit processor, while one might hope that timings would be dominated by the $O(n^2)$ base field multiplication operations, for small values of n the $O(n)$ contribution of the numerous base field additions and subtractions becomes significant, as also observed by Gaudry and Thomé [17]. Observe that on the 64-bit processor a 128-bit field element requires just $n = 2$ (and indeed the description as "multi-precision" should really give way to "double precision"). Therefore it is to be expected that the speed-up we can achieve in this case will be less than might have been hoped.

So is our new method faster? There is really only one satisfactory way to resolve the issue – and that is to identify the fastest known $E(\mathbb{F}_p)$ implementation on a 64-bit processor for the same level of security, and try to improve on it. We understand that the current record is that announced by Gaudry and Thomé at SPEED 2007 [17], using an implementation of Bernstein's `curve25519` [4]. This record is in the setting of an implementation of the elliptic curve Diffie-Hellman method, which requires a single point multiplication to determine the shared secret key.

We point out that the clever implementation and optimizations of `curve25519` are for the sole context of an efficient Diffie-Hellman implementation – ours is general purpose and immediately applicable to a wide range of ECC protocols. In particular the implementation of `curve25519` uses Montgomery's parameterisation of an elliptic curve, is not required to maintain a y coordinate, and hence can achieve compression of the public key at no extra cost (i.e., without the calculation of a square root).

On the other hand we have the use of a particularly nice modulus $2^{127} - 1$, which brings many benefits. For example a base field square root of a quadratic residue x can be calculated as simply $x^{2^{125}}$.

In order to be competitive we wrote a specialised hand-crafted x86-64 assembly language module to handle the base field arithmetic, and integrated this with the MIRACL library. Given that each field element can be stored in just two 64-bit registers, this code is quite short, and did not take long to generate, optimize and test.

Table 3. Point multiplication timings – 64-bit processor

Intel Core 2 processor	Method	Clock cycles
$E(\mathbb{F}_p)$, 255-bit p	Montgomery [17]	386,000
$E(\mathbb{F}_{p^2})$, 127-bit p	SSW	490,000
$E(\mathbb{F}_{p^2})$, 127-bit p	GLV+JSF	359,000
$E(\mathbb{F}_{p^2})$, 127-bit p	GLV+INT	326,000

To obtain our timings we follow Gaudry and Thomé, and utilise two different methods, one based on actual cycle counts, and a method which uses an operating system timer. There are problems with both methods [17], so here we average the two. In practise the two methods were in close agreement, but not of sufficient accuracy to justify exact numbers – so we round to the nearest 1000 cycles. See table 3 for our results. As can be seen, our best method takes 0.84 of the time of the Gaudry and Thomé implementation. Note that point decompression, as required by a Diffie-Hellman implementation which wishes to minimise the size of the public key using point decompression, would require approximately an extra 26,000 clock cycles for our implementation.

It is interesting to observe from table 3 that a careful implementation over a quadratic extension which does not exploit our homomorphism is substantially slower, taking 490,000 cycles. So again it seems that merely switching to a smaller field size is not by itself advantageous on a 64-bit processor, although some of the difference can be explained by the particularly clever parameterization chosen for `curve25519`. However by using our homomorphism we are still able to make up this difference, and indeed overtake the previous record.

To ensure a fair comparison, we also utilise the very useful and easy to use `eBats` facility [10]. Our eBat implements a Diffie-Hellman key exchange algorithm, and can be directly and independently compared with an implementation based on `curve25519`. There are two main functions for a Diffie-Hellman implementation, one which calculates the key pair, and a second which calculates the shared secret. For the key pair calculation we exploit the fact that for our method a multiplication of a fixed point can benefit from extensive off-line precomputation, and use a fixed-base comb algorithm [19], and so this calculation requires only 146,000 cycles. For the shared secret calculation we use the GLV+INT method, plus the cost of a point decompression.

Our latest eBat can be downloaded from `ftp://ftp.computing.dcu.ie/pub/crypto/gls1271-3.tar`. Profiling the code reveals that our with-compression version spends 49% of its time doing base field multiplications and squarings, 15.34% of the time doing base field additions and subtractions and nearly 6% of the time is required for the few modular inversions.

8.5 ECDSA/Schnorr Signature Verification

Verification of both ECDSA and Schnorr signatures requires the calculation of $[a]P + [b]Q$, where P is fixed. In our setting we must calculate $[a_0]P + [a_0]\psi(P) + [b_0]Q + [b_1]\psi(Q)$ – in other words a 4-dimensional multiexponentiation algorithm is required.

Again we use an interleaving algorithm, using windows over a NAF expansion. Since P is now fixed, precomputation of multiples of P (and therefore of $\psi(P)$) can be carried out offline, and so a larger window size of 6 can be used for the multiplication of P. This requires the precomputation and storage of 42 points. For the online precomputation required on Q, we again use a window size of 4 over NAF expansions.

Table 4. Signature Verification timings – 64-bit processor

Intel Core 2 processor	Method	$\mathbb{F}_p$ muls	$\mathbb{F}_p$ adds/subs	Clock cycles
$E(\mathbb{F}_{p^2})$, 127-bit p	GLV+INT	5174	12352	425,000
$E(\mathbb{F}_{p^2})$, 127-bit p	INT	7638	19046	581,000

In table 4 we compare our method with a method that does not use our homomorphism while using the 2-dimensional interleaving algorithm which calculates $[a]P + [b]Q$ directly for random $a, b < r$, again using a size 6 window for the fixed point P, and a size 5 window for the variable point Q.

Antipa et al [1] propose a variant of ECDSA with faster signature verification (note that their method does not apply to Schnorr signatures). The basic method gives essentially the same performance as our method (they transform $[a]P + [b]Q$ to a 4-dimensional multiexponentiation with coefficients $\approx \sqrt{r}$). Their method, as with ours, assumes that P is fixed and that certain precomputation has been done.

The paper [1] also gives a variant where the public key is doubled in size to include Q and $Q_1 = [2^{\lceil \log_2(r)/3 \rceil}]Q$. Their method transforms $[a]P + [b]Q$ to a 6-dimensional multiexponentiation with coefficients of size $\approx r^{1/3}$. In this context (i.e., enlarged public keys) we can improve upon their result. Let $M = 2^{\lceil \log_2(r)/4 \rceil}$ and suppose the public key features Q and $Q_1 = [M]Q$. The GLV idea transforms $[a]P + [b]Q$ to $[a_0]P + [a_1]\psi(P) + [b_0]Q + [b_1]\psi(Q)$ where $a_0, a_1, b_0, b_1 \approx \sqrt{r}$. We now write $a_0 = a_{0,0} + Ma_{0,1}$ where $a_{0,0}, a_{0,1} \approx r^{1/4}$ and similarly for a_1, b_0, b_1. Hence the computation becomes an 8-dimensional multiexponentiation with coefficients of size $\approx r^{1/4}$. Another advantage of our method is that it applies to Schnorr signatures whereas the method of [1] is only for ECDSA and other variants of ElGamal signatures.

9 Security Implications

The homomorphism ψ of Theorem 1 (at least, in the case when ϕ is an isomorphism) defines equivalence classes of points in $E'(\mathbb{F}_{p^m})$ of size $2m$ by $[P] = \{\pm\psi^i(P) : 0 \leq i < m\}$. By the methods of Gallant-Lambert-Vanstone [14] and Wiener-Zuccherato [34] one can perform the Pollard algorithms for the discrete logarithm problem on these equivalence classes. This speeds up the solution of the discrete logarithm problem by a factor of $\sqrt{m}$ over previous techniques. When $m = 2$ then this is not a significant loss of security.

A more serious threat to our proposal comes from the Weil descent philosophy, and in particular the work of Gaudry [16]. Gaudry gives an algorithm for the discrete logarithm problem in $E'(\mathbb{F}_{p^m})$ requiring time $O(p^{2-4/(2m+1)})$ group operations (with bad constants) which, in principle, beats the Pollard methods for $m \geq 3$. Our proposal in the case $m = 2$ is immune to this attack.

Gaudry's method also applies to abelian varieties: if A is an abelian varitey of dimension d over $\mathbb{F}_{p^m}$ then the algorithm has complexity $O(p^{2-4/(2dm+1)})$. Hence, for Jacobians of genus 2 curves over $\mathbb{F}_{p^2}$ one has an algorithm running in time $O(p^{1.55})$, rather than the Pollard complexity of $O(p^2)$. Gaudry's method is exponential time and so one can secure against it by increasing the field size. For example, to achieve 128-bit security level using our construction with genus 2 curves over $\mathbb{F}_{p^2}$ or elliptic curves over $\mathbb{F}_{p^4}$ one should take p to be approximately 80 bits rather than the desired 64 bits.

Acknowledgements

We thank Dan Bernstein, Billy Brumley, Jinhui Chao, Pierrick Gaudry, Darrel Hankerson, Alfred Menezes, Yasuyuki Nogami and Fre Vercauteren for suggestions and comments.

References

1. A. Antipa, D. R. L. Brown, R. P. Gallant, R. J. Lambert, R. Struik and S. A. Vanstone, Accelerated Verification of ECDSA Signatures, in B. Preneel and S. E. Tavares, SAC 2005, Springer LNCS 3879 (2006) 307-318.
2. R. Avanzi, Aspects of Hyperelliptic Curves over Large Prime Fields in Software Implementations, CHES 2004, Springer LNCS 3156 (2004), 148–162
3. R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen and F. Vercauteren, Handbook of elliptic and hyperelliptic cryptography, Chapman and Hall/CRC, (2006).
4. D. J. Bernstein, Curve25519: New Diffie-Hellman Speed Records, in M. Yung et al (eds), PKC 2006, Springer LNCS 3958 (2006) 207–228.
5. D. J. Bernstein, Differential addition chains, (2006). preprint.
6. D. J. Bernstein, Elliptic vs. hyperelliptic, part 1 ECC 2006, Toronto, Canada http://www.cacr.math.uwaterloo.ca/conferences/2006/ecc2006/slides.html
7. D. J. Bernstein and T. Lange, Faster addition and doubling on elliptic curves, in K. Kurosawa (ed), Asiacrypt 2007, Springer LNCS 4833 (2007) 29–50.
8. D. J. Bernstein and T. Lange, Inverted Edwards coordinates, in S. Boztas and H.-F. Lu (eds.), AAECC 2007, Springer LNCS 4851 (2007) 20–27.
9. D. J. Bernstein and T. Lange, Analysis and optimization of elliptic-curve single-scalar multiplication, Finite fields and applications: Proceedings of Fq8, Contemporary Mathematics 461, American Mathematical Society, (2008) 1–18
10. eBATS: ECRYPT Benchmarking of Asymmetric Systems, http://www.ecrypt.eu.org/ebats/
11. D. R. L. Brown, Multi-Dimensional Montgomery Ladders for Elliptic Curves, eprint 2006/220.
12. E. Dahmen, K. Okeya and D. Schepers, Affine Precomputation with Sole Inversion in Elliptic Curve Cryptography, in J. Pieprzyk, H. Ghodosi and E. Dawson (eds.), ACISP 2007, Springer LNCS 4586 (2007) 245-258.
13. S. D. Galbraith and M. Scott, Exponentiation in pairing-friendly groups using homomorphisms, in S. D. Galbraith and K. G. Paterson (eds.), Pairing 2008, Springer LNCS 5209 (2008).
14. R. P. Gallant, R. J. Lambert and S. A. Vanstone, Improving the parallelized Pollard lambda search on anomalous binary curves, *Math. Comp.*, **69** (2000), 1699-1705.

15. R. P. Gallant, R. J. Lambert and S. A. Vanstone, Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In J. Kilian (Ed.), CRYPTO 2001, Springer LNCS 2139 (2001), 190–200.
16. P. Gaudry, Index calculus for abelian varieties and the elliptic curve discrete logarithm problem, to appear in *J. Symbolic Comput.*
17. P. Gaudry and E. Thome, The mpFq library and implementing curve-based key exchanges. SPEED workshop presentation, Amsterdam, June 2007
18. P. J. Gabner,C. Heuberger and H. Prodinger, Distribution results for low-weight binary representations for pairs of integers, *Theoretical Comp. Sci.*, **319** (2004) 307–331.
19. D. Hankerson, A. J. Menezes and S. Vanstone, Guide to elliptic curve cryptography, Springer (2004).
20. D. Hankerson, K. Karabina and A. J. Menezes, Analyzing the Galbraith-Lin-Scott Point Multiplication Method for Elliptic Curves over Binary Fields, eprint 2008/334.
21. T. Iijima, K. Matsuo, J. Chao and S. Tsujii, Costruction of Frobenius maps of twists elliptic curves and its application to elliptic scalar multiplication, SCIS 2002.
22. S. Kozaki, K. Matsuo, and Y. Shimbara, Skew-Frobenius maps on hyperelliptic curves, IEICE Trans. E91-A (2008)
23. A. J. Menezes, Another look at HMQV, *J. Mathematical Cryptology*, 1 (2007), 47-64.
24. B. Möller, Algorithms for multi-exponentiation. In S. Vaudenay and A. M. Youssef (Eds.), SAC 2001, Springer LNCS 2259 (2001), 165–180.
25. B. Möller, Improved Techniques for Fast Exponentiation. In P. Lee and C. Lim (Eds.), ICISC 2002, Springer LNCS 2587 (2003), 298–312.
26. Y. Nogami and Y. Morikawa, Fast generation of elliptic curves with prime order over $\mathbb{F}_{p^{2^c}}$, in Proceedings International Symposium on Information Theory 2003.
27. J. Proos, Joint sparse forms and generating zero solumns when combing, Technical report CORR 2003-23, CACR (2003)
28. M.Scott, MIRACL – Multiprecision Integer and Rational Arithmetic C/C++ Library, http://ftp.computing.dcu.ie/pub/crypto/miracl.zip, 2008
29. M. Scott and P. Szczechowiak, Optimizing Multiprecision Multiplication for Public Key Cryptography, preprint 2007. http://eprint.iacr.org/2007/299
30. F. Sica, M. Ciet, J.-J. Quisquater, Analysis of the Gallant-Lambert-Vanstone method based on efficient endomorphisms: Elliptic and hyperelliptic curves, in K. Nyberg and H. M. Heys (eds.), SAC 2002, Springer LNCS 2595 (2003) 21–36.
31. J. H. Silverman. *The Arithmetic of Elliptic Curves.* Graduate Texts in Mathematics 106. Springer-Verlag, 1986.
32. J. A. Solinas, Low-weight binary representations for pairs of integers, Technical report CORR 2001–41, CACR, 2001.
33. M. Stam, Speeding up Subgroup Cryptosystems, Ph.D. thesis, Technische Universiteit Eindhoven, 2003.
34. M. J. Wiener and R. J. Zuccherato, Faster Attacks on Elliptic Curve Cryptosystems. In S. Tavares and H. Meijer (Eds.), SAC 1998, Springer LNCS 1556 (1999), 190–200.

A Joint Sparse Forms

Solinas [32] proposed the joint sparse form (JSF) for double exponentiation which has a joint Hamming weight approximately 1/2 the bitlength of the pair of exponents. Write $l(n) = \lceil \log_2(n) \rceil$. If a point multiplication $[n]P$ is performed using a non-adjacent form representation of n then it requires $l(n)$ doubles and, on average, $l(n)/3$ additions. On the other hand, using a GLV method to compute $[n]P$ using multiexponentiation and the Solinas JSF requires $l(n)/2$ doubles and $l(n)/4$ additions. This makes the multiexponentiation significantly faster (ignoring precomputation) than the single exponentiation when we are using simple signed expansions.

However, the above analysis fails to account for the fact than in practice one uses window methods and precomputation to compute $[n]P$, and that doublings are in general faster to compute than additions. As discussed in Section 7, one finds that interleaving methods are simpler and more useful than using joint sparse forms.

When considering m-dimensional versions of the GLV method or signature verification then one could consider higher-dimensional variants of the joint sparse form. Grabner, Heuberger and Prodinger [18] and Proos [27] have considered this problem and given algorithms to construct such representations for any m. However, the performance improvement becomes small as m grows. Roughly, one expects to get a joint signed representation of the m exponents n_i such that each block of $m+1$ bits has two 'zero columns'. In other words, using a generalised JSF one expects the multiexponentiation to cost $1 + l(n)/m$ doubles plus $(m-1)/(m+1)(1 + l(n)/m)$ adds, compared with $l(n)/m$ doubles plus $log_2(n)/m$ adds for the naive method. In practice, it seems not useful to use joint sparse forms for this application, and we instead suggest using interleaving and "windowing the NAF".

B More About Multiexponentiation

As noted in the main text, there are a number of methods for multiexponentiation in the literature (see [3, 19] for surveys). For example, one can compute a joint sparse form and then do the standard multiexponentation (Algorithm 3.48 of [19]). Alternatively, one can compute an ω-NAF for some window size ω and perform interleaving (Algorithm 3.51 of [19]) or, as we propose, compute a NAF and then use the interleaving method by taking sliding windows of length ω over the NAF.

The following table gives the results of simulations of these methods. For each method the number of doubles is fixed, so the interesting part is the average number of adds per bit. Let D, A, and E represent the doubling, addition, and the homomorphism respectively. Note that the precomputation and storage costs are estimated using standard elliptic curve arithmetics. In the implementation, the strategy of Dahmen, Okeya and Schepers (DOS) [12] can be used, this will not significantly change the relative comparison.

Table 5. Efficiency comparision for computing $[m]P$

Method	Cost of each bit	Precomputation	Storage
Simultaneous($\omega = 2$)	1D + 0.5A	11A + 3E	15 points
JSF	1D + 0.5A	2A + E	4 points
Interleaving width-4-NAF	1D + 0.407A	3A + 1D + 4E	8 points
Interleaving width-5-NAF	1D + 0.342A	7A + 1D + 8E	16 points
Interleaving ω-SW- NAF ($\omega = 4$)	1D + 0.385A	4A + 1D + 5E	10 points
Interleaving ω-SW- NAF ($\omega = 5$)	1D + 0.318A	10A + 1D + 11E	22 points

For 128-bit exponents Interleaving using 4-NAF, Interleaving using 5-NAF, and our variant all require about the same number of adds. The advantage of our approach is that the set-up costs (e.g., computing the NAF) seem to be cheaper.

C Point multiplication in Montgomery form

Bernstein [5] and Brown [11] have discussed the analogue of multiexponentiation in the case when only x-coordinates are used (for example, using Montgomery elliptic curves). Using their results one can implement the GLV method in this manner.

Unlike the standard case things are not so simple. In particular, y-coordinates are needed to compute the initial steps. Hence, one cannot avoid "point decompression" if one wants to use the GLV method.

Theoretically, it is attractive to use the Montgomery Euclidean ladder (see Section 3.3 of [33]). This method computes $[m]P + [n]\psi(P)$, where the addition and doubling are done using Montgomery's method. Heuristically, it costs about $1.49A + 0.33D$ for each bit of the exponent (Conjecture 3.30 of [33]). However, this method requires reduction modulo 3 and division by 3 as part of the control code. These operations are difficult to optimise and in our implementation the control code required half of the computation time. Hence, despite the theoretical beauty of this method, it does not seem to be appropriate for practical applications.