ABSTRACT

Title of Thesis: FAST HASHING INTO ELLIPTIC AND HYPERELLIPTIC CURVES:

Michael Goldman, Master of Arts, 2011

Thesis directed by: Professor Lawrence C. Washington
Department of Mathematics

The BLS Digital Signature Algorithm is a cryptographic scheme using elliptic curves over finite fields. In the BLS Digital Signature Algorithm, we require a function that maps a arbitrary element of a finite field to an elliptic curve in that field. There is a simple, naive way to do this, but it is not particularly fast.

In a recent paper, Paulo S. L. M. Barreto and Hae Y. Kim described an algorithm that speeds up the computation significantly (by about two orders of magnitude). This speed up in computation is due to avoiding square roots in these finite fields. Square root computation in these fields leads to speeds of $O(r^3)$. Barreto and Kim's techniques uses the fact that cubing in a finite field of characteristic 3 is a fast, linear operation on the order of $O(r)$. Barreto and Kim's complete algorithm takes $O(r^2)$ steps, a major improvement.

However, this algorithm is only for specific elliptic curves of the form $x^3 - x + b = y^2$. The goal of this paper is to extend this algorithm to significantly more elliptic curves in characteristic 3. To do so, we consider other elliptic curves

of similar construction (of the form $x^3 + x + b = y^2$), and then we reduce much more general cases to these two curves. We also describe the limitations of these techniques for other elliptic curves.

We then extend this algorithm to a more general case of hyperelliptic curves in characteristic p. Similarly to the case in characteristic 3, raising to the pth power is a fast operation in characteristic p. We first look at a more simple case, where the elliptic curve is described as $x^p + \beta x + b = y^2$, where β is in the base field of $\mathbb{F}_p$. Then we seek to generalize to where β is any element in our field. Once again, we describe the limitations of these techniques when applying them to other curves.

After initial setup costs, the algorithm remains at the speed $O(r^2)$. In certain cases, the algorithm will converge at a rate $1/p$, and in other cases, the algorithm will be deterministic.

FAST HASHING INTO ELLIPTIC AND HYPERELLIPTIC CURVES

by

Michael Goldman

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Arts
2011

Dedication

This mathematics dissertation is dedicated to my father, Professor William Goldman, who continues to inspire me as a mathematician and a friend.

Table of Contents

List of Abbreviations

α	alpha
β	beta
γ	gamma
ω	omega
$\mathbb{F}_q$	field of q elements
BLS	Boneh-Lynn-Shacham

1 Introduction

In recent years, the development of elliptic curve cryptography has given rise to much more efficient cryptographic schemes, such as the digital signing scheme by Boneh, Lynn, and Shacham also known as the BLS Signature Algorithm. Unlike other previous schemes, this scheme has no equivalent with more conventional groups like $\mathbb{Z}_p$. While other kinds of signature algorithms are known and can produce smaller signatures, the BLS Signature is significantly faster in practice.

In the BLS Signature Algorithm, there is a specific need for a function to map from $\mathbb{F}_q$ to the elliptic curve in question. The simplest function has a running time of $O(r^3)$, and is restricted by the need to calculate square roots. Paulo S. L. M. Barreto and Hae Y. Kim discussed a fast hash into an elliptic curve of characteristic three [1]. By avoiding square root evaluations, they developed a map that takes $O(r^2)$ steps.

This algorithm can be found in Section 2 and is the basis of this paper. While Barreto and Kim's paper discuss the properties of this algorithm on a specific elliptic curve, this paper uses their techniques to extend their algorithm to other curves. Not only do we extend this method to more elliptic curves in characteristic three, but we also extend their methods to hyperelliptic curves in characteristic p.

1.1 The BLS Signature Scheme

The BLS Signature Algorithm is the motivating factor for this paper. The digital signature algorithm goes as follows:

The BLS Signature Algorithm

1. Alice sets up the signature scheme. She chooses a public elliptic curve $E \in \mathbb{F}_{p^r} \times \mathbb{F}_{p^r}$, and a public elliptic curve point G. Alice chooses a *Pairing Operation* $\rho : E \times E \to \mathbb{F}_q^\times$. This pairing operation is bilinear and one-way. She chooses a mapping $H(y) : \mathbb{F}_q^\times \to E$

2. Alice has a private key k, and a message m that she would like to sign. She lets $\hat{G} = k \cdot G$, and makes $\hat{G}$ public.

3. Alice signs a message m. She lets $S = k \cdot H(m)$ and she sends the pair (S, m) to Bob.

4. Bob verifies the signature. He checks that $\rho[S, G] = \rho[H(m), \hat{G}]$, which works simply by the bilinear property of ρ.

In the above and in the following, we do not need E to be the full group of points on the elliptic curve with coordinates in $\mathbb{F}_{p^r}$. A subgroup suffices, and we denote such a subgroup also by E. This paper deals with $H(y)$, the function that hashes onto the elliptic curve E. The goal is to find methods for incredibly fast hash functions.

First of all, one may wonder why we even need $H(y)$. A naive way to do this scheme is to instead sign a message with $S = k \cdot h(m) \cdot G$, where $h(m)$ is an integer valued hash function. This seems intuitive because we already use a hash function. However, this makes it possible to forge a signature on any message. As the hash function is public information, anyone could simply send $(h(m) \cdot \hat{G}, m)$ and it would satisfy the verification.

Therefore, we need some way to map to points on the elliptic curve *without* the discrete logarithm of the signature with respect to G being known.

1.2 Standard Method

Here we present the standard, simplest method of mapping to the elliptic curve. Start with an elliptic curve in the form:

$$x^3 + \alpha x + b = y^2 \tag{1.1}$$

The Standard Mapping Algorithm [1]

1. Choose $x \in \mathbb{F}_q$.
2. Check if $x^3 + \alpha x + b$ is a square in the field.
3. If it is, take the square root. Otherwise, let x be the next x (according to some ordering) and go back to step 1.

This is obviously a very simple and direct way to obtain a point on the elliptic curve. As half of the numbers are squares, it is a 1/2 chance that you will obtain a point immediately. The main issue with this computation is taking a square root in a finite field.

Taking square roots is not a particularly fast operation (on the order of $O(r^3)$), and can slow down the computation. To perform this action, one would need to calculate a square root for every choice of x. In Section 2, we give a method that is significantly faster.

We choose x by hashing our message with a typical integer hash function. Henceforth, when we say "we choose x or y", we imply that we hash our message into a value in $\mathbb{F}_{p^r}$.

1.3 Normal Bases

There are many choices we have for the field $\mathbb{F}_q$. The two main ones that are used for security purposes are when q is a large prime, and when $q = p^r$, where p is a small prime, and r is a large prime. We say that $\mathbb{F}_{p^r}$ is a *field of characteristic* p. These fields have specific properties related to their characteristic.

The main advantage of working in $\mathbb{F}_{p^r}$ is that it can broken down into a vector space of dimension r, where each coordinate is represented by an element in $\mathbb{F}_p$. The computation can be done in $\mathbb{F}_p$ which has relatively fast computation. A standard representation of $\mathbb{F}_{p^r}$ is $\mathbb{F}_p[x]/f(x)$ where f is an irreducible polynomial of degree r. This yields the basis $\{1, x, x^2, \cdots, x^{r-1}\}$. However, for our purposes it is better to work with a *normal* basis. A normal basis has several properties that we want.

For any basis, if $a_0, a_1, ...a_{r-1}$ are our basis elements, then all $n \in \mathbb{F}_{p^r}$ have unique representions $n = \sum_{k=0}^{r-1} n_k a_k$ where each $n_k \in \mathbb{F}_p$. This gives us the vector space. But there is an additional property of the normal basis; namely, $a_k^p = a_{k+1}$. This means that each element of the normal basis can be raised to the pth power to shift to the next basis element. Also, $n_k \in \mathbb{F}_p$, so $n_k^p = n_k$ by Fermat. Henceforth, consider all indices to be taken $\mod r$ except where specifically denoted otherwise.

The normal basis yields a powerful formula with exponents: if

$$n = \sum_{k=0}^{r-1} n_k a_k \tag{1.2}$$

then we can express n^p in the following way:

$$n^p = \sum_{k=0}^{r-1} a_k n_{k+1} \tag{1.3}$$

Simply put, raising something to the pth power in a field of characteristic p shifts the coefficients of the normal basis over one. That means raising to the pth power is a linear operation.

Consider the element $a_0 + a_1 + \cdots + a_{r-1}$. Notice that when you raise this to the pth power, it is itself. So this is an element of $\mathbb{F}_p$. In fact by rescaling by a suitable element of $\mathbb{F}_{p^r}$, we may assume $a_0 + a_1 + \cdots + a_{r-1} = 1$, and similarly $-a_0 - a_1 - \cdots - a_{r-1}$ to be -1. Also, if $\beta \in \mathbb{F}_p$, then it is represented as $\beta a_0 + \beta a_1 + \cdots + \beta a_{r-1} \in \mathbb{F}_{p^r}$

Now that we have described the properties of the normal basis, how do we find this normal basis?

1.4 Finding a Normal Basis

We represent $\mathbb{F}_{p^r}$ by $\mathbb{F}_p[x]/f(x)$ where $f(x)$ is some monic polynomial of degree r. We define the gcd of two polynomials to be the monic polynomial of highest degree that divides both polynomials. We present an algorithm for finding a normal basis using this construction.

Probablistic Normal Basis Search [3]

1. Choose $a \in \mathbb{F}_{p^r}$ at random
2. Compute a^{p^i} for $1 \leq i \leq r$
3. Let $\omega(x) = \sum_{i=1}^{r} a^{p^i} x^i$. Calculate the $\gcd(\omega(x), x^r - 1)$.
4. If $\gcd(\omega(x), x^r - 1) = 1$ then $\{a^{p^i}\}$ is a normal basis. If not, try the algorithm again.

Clearly the elements of $\{a^{p^i}\}$ are shifted by the pth-power map. The question is why does the gcd check that this is a basis?

Theorem *(Hensel)* $\gcd(\omega(x), x^r - 1) = 1$ if and only if $\{a^{p^i}\}$ is a basis.

Proof. Suppose:

$$x_0a_0 + x_1a_1 + \cdots + x_{r-1}a_{r-1} = 0$$

with some $\{x_k\} \in \mathbb{F}_p$. Let $\omega(x)$ be the function to the left. We apply the pth power map several times, along with $a_i^p = a_{i+1}$ and $x_i^p = x_i$ to obtain:

$$\begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{r-1} \\ a_{r-1} & a_0 & a_1 & \cdots & a_{r-2} \\ a_{r-2} & a_{r-1} & a_0 & \cdots & a_{r-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & a_3 & \cdots & a_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{r-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

We let the matrix on the left be matrix D. First we will show that if $\{a^{p^i}\}$ has a nonzero solution with entries in $\mathbb{F}_{p^r}$ then there is a nonzero solution with entries in $\mathbb{F}_p$ if these rows are independent as vectors. Suppose $\{x_i\}$ is in the nullspace of the matrix. Looking at the $r-j+1$st row for any j yields the relation:

$$x_1a_{r-j+1} + x_2a_{r-j+2} + \cdots + x_ra_{r-j} = 0$$

Now apply the p^jth power map to this to obtain:

$$x_1^{p^j}a_{r+1} + x_2^{p^j}a_{r+2} + \cdots + x_r^{p^j}a_r = 0$$

It is easy to see that $(x_1^{p^j}, x_2^{p^j}, \cdots, x_{r-1}^{p^j})$ is in the nullspace of our matrix for any j. Therefore the sum of all these elements is also in the null space. But $x_i + x_i^p + x_i^{p^2} + \cdots + x_i^{p^r-1}$ is equal to its pth power, so it is in $\mathbb{F}_p$. We assume x_i is nonzero, so we can rescale it to be 1. As $\gcd(r,p) = 1$ because r is a large prime, we have $x_i + x_i^p + x_i^{p^2} + \cdots + x_i^{p^r-1} = r \neq 0$. So we have a nonzero vector with entries in $\mathbb{F}_p$ in the nullspace. Therefore if we have a solution with entries in $\mathbb{F}_{p^r}$ then we have a solution with entries in $\mathbb{F}_p$.

Now, we want the $\{a^{p^i}\}$ to be independent as vectors, which implies that $\det D \neq 0$. D is called a *circulant matrix*, and it has a well known determinant. Let ζ be an r^{th} root of unity in the algebraic closure of our field. We have:

$$\begin{pmatrix} a_0 & a_1 & a_2 & \cdots & a_{r-1} \\ a_{r-1} & a_0 & a_1 & \cdots & a_{r-2} \\ a_{r-2} & a_{r-1} & a_0 & \cdots & a_{r-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & a_3 & \cdots & a_0 \end{pmatrix} \begin{pmatrix} 1 \\ \zeta \\ \zeta^2 \\ \vdots \\ \zeta^{r-1} \end{pmatrix} = (a_0 + a_1\zeta + \cdots + a_{r-1}\zeta^{r-1}) \begin{pmatrix} 1 \\ \zeta \\ \zeta^2 \\ \vdots \\ \zeta^{r-1} \end{pmatrix}$$

Therefore $v_\zeta = (1, \zeta, \zeta^2, \cdots, \zeta^{r-1})$ is an eigenvector with eigenvalue $a_0 + a_1\zeta + \cdots + a_{r-1}\zeta^{r-1}$, for all ζ such that $\zeta^r = 1$. The vectors $(1, \zeta, \zeta^2, \cdots, \zeta^{r-1})$ as ζ runs through all roots of $x^r - 1$ are linearly independent (as they form a Vandermonde matrix with determinant nonzero), so we have r eigenvectors. Therefore, this yields the determinant of the matrix D:

$$\det \begin{pmatrix} a_0 & a_1 & a_2 & \cdots & a_{r-1} \\ a_{r-1} & a_0 & a_1 & \cdots & a_{r-2} \\ a_{r-2} & a_{r-1} & a_0 & \cdots & a_{r-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & a_3 & \cdots & a_0 \end{pmatrix} = \prod_{\zeta^r=1} (a_0 + \zeta a_1 + \zeta^2 a_2 + \cdots + \zeta^{r-1} a_{r-1})$$

In order for this product to be zero, there must be a root of unity ζ such that:

$$a_0 + \zeta a_1 + \zeta^2 a_2 + \cdots + \zeta^{r-1} a_{r-1} = 0$$

which means ζ is a zero of $\omega(x)$. By definition, ζ is a zero of $x^r - 1$, so $\gcd(\omega(x), x^r - 1) \neq 1$. So $\{a_i\}$ are linearly dependent if and only if there is a nonzero solution to D if and only if $\gcd(\omega(x), x^r - 1 = 0) \neq 1$.

Therefore the Probabilistic Normal Basis Search obtains a normal basis.

2 The Main Algorithm

We now introduce the main algorithm of the paper. This algorithm was introduced by Paulo S. L. M. Barreto and Hae Y. Kim in their paper "Fast Hashing Onto Elliptic Curves Over Fields of Characteristic 3." It avoids taking square roots in these finite fields entirely. The Standard Mapping Algorithm chooses x and solves the square root for y. This algorithm works in the reverse. We choose y and solve for x to avoid the square root.

This algorithm works specifically with a field of characteristic three and the elliptic curve, $x^3 - x + b = y^2$. It uses the fact that cubing is a linear operation in this finite field. With this, we speed up the computation.

Algorithm 1

1. Begin with a normal basis in the field of 3^r elements where r is prime. The basis elements will be denoted $a_0, a_1, \cdots, a_{r-1}$. Let E be an elliptic curve over $\mathbb{F}_{3^r}$ with the equation:

$$x^3 - x + b = y^2$$

2. Choose $y \in \mathbb{F}_{3^r}$ and let $u = y^2 - b$. So we have the equation:

$$x^3 - x = u \tag{2.1}$$

3. Let u_k be the coefficients of u:

$$u = u_0 a_0 + u_1 a_1 + \cdots + u_{r-1} a_{r-1}$$

4. Let x_0 to be a fixed element of $\mathbb{F}_3$ (for example, let $x_0 = 0$). Run through the series of computations to obtain $\{x_k\}$:

$$\begin{aligned} x_1 &= u_0 + x_0 \\ x_2 &= u_1 + x_1 \\ &\cdots \\ x_r &= u_{r-1} + x_{r-1} \end{aligned}$$

5. Now check if $x_r = x_0$. If so, let $x = x_0 a_0 + x_1 a_1 + \cdots + x_{r-1} a_{r-1}$, and (x, y) is a point on the elliptic curve. If not, then that choice of y does not yield a point on the curve. In this case, use the next value of y and go back to step 1.

After initial setup, this algorithm takes $O(r^2)$ time. While our techniques allowed Step 4 to be $O(r)$ time, choosing y and computing u ends up dominating the entire algorithm. Not only does Step 4 only contain $O(r)$ operations, it only involves addition in the field $\mathbb{F}_3$, so the computation is incredibly fast. Why does this yield a point on the curve?

Cubing in a field of characteristic three is a linear operation. In terms of the normal basis, if $x = \sum_{k=0}^{r-1} x_k \cdot a_k$ then $x^3 = \sum_{k=0}^{r-1} x_{k+1} a_k$. So solving the equation $x^3 - x = u$ involves solving the equation for each of the coefficients of the basis elements, which are all elements of F_3.

In Step 4, we have that the map $x \to x^3 - x$ has kernel of 3 elements: 0, and $\pm 1 = \pm(a_0 + \cdots + a_{r-1})$. So if any choice of x_0 fails, then all three choices of x_0 will fail. So our only option is pick an entirely different y.

This algorithm is the basis of this paper. We will see how this technique works in different elliptic curves in fields of characteristic three, and how we can stretch this algorithm to hyperelliptic curves in characteristic p.

2.1 Speed Comparison and Timings

The following table compares the timing of the Standard Mapping Algorithm with Algorithm 1. The algorithms were coded in C++ and run on a 550 Mhz Pentium III Processor, and these timings are taken directly from Barreto and Kim's paper on this algorithm [1].

Running Times of Algorithms in ms

r	Standard Mapping	Algorithm 1
79	3.02	0.037
97	5.53	0.054
163	25.7	0.164

As one can see, Algorithm 1 noticeably faster. It speeds up the algorithm by about two orders of magnitude. This technique is neither limited to this elliptic curve nor to this characteristic, and this paper explores these other possibilities.

3 Extending to other Curves

Our goal is to extend Algorithm 1 to all equations of the form $x^3 + \alpha x = u$. However, before we do that, we need to consider another particular elliptic curve.

3.1 The Other Base Case

Consider the elliptic curve:

$$y^2 = x^3 + x + b$$

We now present a similar algorithm to map onto this curve:

Algorithm 2

1. Begin with a normal basis in the field of 3^r elements where r is prime. The basis elements will be denoted a_1, a_2, $\cdots$, a_{r-1}. Let E be an elliptic curve over $\mathbb{F}_{3^r}$ with the equation:

$$x^3 + x + b = y^2$$

2. Choose $y \in \mathbb{F}_{3^r}$. Let $u = y^2 - b$ as before, yielding the equation:

$$x^3 + x = u$$

3. Compute the vector x by the following series of computations:

$$\begin{aligned} x_1 &= u_0 - x_0 \\ x_2 &= u_1 - x_1 \\ &\dots \\ x_r &= u_{r-1} - x_{r-1} \end{aligned}$$

4. Check if $x_r = x_0$. If so, then (x, y) is a point on the curve and you're finished.

5. If not, let $e = x_r - x_0$. Compute $\hat{x}$ by the series of computations:

$$\begin{aligned} \hat{x}_0 &= x_0 + e \\ \hat{x}_1 &= x_1 - e \\ &\dots \\ \hat{x}_k &= x_k + (-1)^k e \\ &\dots \\ \hat{x}_{r-1} &= x_{r-1} + e \end{aligned}$$

   and $(\hat{x}, y)$ is a point on the curve.

Algorithm 2 is extremely similar to Algorithm 1. There is a parity change in Step 3, but that is obviously because of the parity change in the original equation. The significant difference comes in at Step 5. We now look at Step 5, and see why this computation gives us a point on the curve. This has the same complexity as Algorithm 1, but it is deterministic.

3.2 Error Term Computation

With $x^3 + x = u$, rather than giving up on a specific y, we can fix x_0 to be the correct choice. First run through the computation and check whether $x_r = x_0$. If so, then we're done. If not, let $e = x_0 - x_r$. Now we run through the algorithm a second time, using $\hat{x}_0 = x_0 + e$. This yields the computation we ended up with in Step 5 of Algorithm 2:

$$\begin{aligned}
\hat{x}_1 &= u_0 - x_0 - e \\
\hat{x}_1 &= x_1 - e \\
\hat{x}_2 &= u_1 - x_1 + e \\
\hat{x}_2 &= x_2 + e \\
&\ldots \\
\hat{x}_k &= x_k + (-1)^k \cdot e
\end{aligned}$$

Finally, we check if this is correct. We know r is odd because it's assumed to be a large prime, so $\hat{x}_r = x_r - e$.

$$\begin{aligned}
x_0 - x_r &= e \\
x_0 - x_r &= -2e \quad (\text{as } -2 = 1 \in \mathbb{F}_3) \\
x_0 + e &= x_r - e \\
\hat{x}_0 &= \hat{x}_r
\end{aligned}$$

Computationally, the correction can be done as follows: Let z be defined as $z_k = (-1)^k$. Then let $\hat{x} = x + ez$ by vector addition. This is another extremely fast method to fix the original choice of x.

3.3 Limitations of the Error Term

One may also be tempted to try the same method of Error Term Computation on the elliptic curve $x^3 - x + b = y^2$. However, we can prove that this fails. Take the elliptic curve $x^3 - x + b = y^2$. Let $\hat{x}_0 = x_0 + e$ for any choice of e. Run through the algorithm to obtain:

$$\begin{aligned}
\hat{x}_1 &= u_0 + x_0 + e \\
\hat{x}_1 &= x_1 + e \\
\hat{x}_2 &= u_1 + x_1 + e \\
\hat{x}_2 &= x_2 + e \\
&\ldots \\
\hat{x}_k &= x_k + e
\end{aligned}$$

and with this we check if the point is on the curve:

$$\begin{aligned}
\hat{x}_0 &\overset{?}{=} \hat{x}_r \\
x_0 + e &\overset{?}{=} x_r + e \\
x_0 &\neq x_r
\end{aligned}$$

by our assumption of our original point not working. This is because the map $x \to x^3 - x$ from $\mathbb{F}_{3^r} \to \mathbb{F}_{3^r}$ is three-to-one. If one x_0 fails, then all choices of x_0 will fail.

3.4 General Elliptic Curves

Now that we have figured out the two base cases, we can extend to a much more general case. Look at the equation:

$$x^3 + \alpha x + b = y^2 \qquad \text{with } \alpha \in \mathbb{F}_{3^r}.$$

Our method for solving this problem is to find t such that $t^2 = \pm\alpha$. Once we find such a t, we can use either Algorithm 1 or Algorithm 2 to find points quickly on the curve.

Algorithm 3

1. Let E be an elliptic curve onto $\mathbb{F}_{3^r}$ with the equation $x^3 + \alpha x + b = y^2$.
2. Let t be a square root of α or $-\alpha$. By the transformation $x \to tx$, we reduce our equation to $x^3 \pm x = (y^2 - b)/(\alpha t)$.
3. Let $u = \frac{y^2-b}{\alpha t}$ to obtain the equation $x^3 \pm x = u$.
4. Use Algorithm 1 or Algorithm 2 to obtain x.
5. Transform back the solution $x \to x/t$.

The additional computation is calculating the square root of α, division by t^3, and transforming back and forth. However, other than the transformations back and forth, these operations only are computed once for the whole curve, independently of how many points are computed on the curve. So calculating the square root will take $O(r^3)$ operations initially. For every point we want to calculate on the curve, it takes $O(r^2)$ operations to multiply and divide elements.

Remember, we are working a field of 3^r elements where r is prime (and therefore odd), which means that -1 is not a square in the field. So of course if α is not a square, then $-\alpha$ is a square and vice versa.

3.5 Taking the Square Root

Lemma If α is a square in a finite field with 3^r elements, then we can calculate t such that $t^2 = \alpha$ by the following operation:

$$t = \sqrt{\alpha} = \alpha^{\frac{3^r+1}{4}} \tag{3.1}$$

Proof: It should be noted that r is a large prime and therefore odd. Therefore, $3^r = 3 \mod 4$ so we know that the exponent is an integer. Therefore, this operation is well defined. To see why this operation yields the element we are looking for, square the result:

$$\begin{aligned} &\alpha^{\frac{3^r+1}{4}\cdot 2} \\ &= \alpha^{\frac{3^r+1}{2}} \\ &= \alpha^{\frac{3^r-1}{2}} \cdot \alpha \end{aligned}$$

Note that $\alpha^{\frac{3^r-1}{2}}$ is a square root of unity, in this case it is ± 1. This also gives an algorithm for deciding which of $\pm\alpha$ is a square. Compute $(\alpha^{\frac{3^r+1}{4}})^2$. It will be either α or $-\alpha$, and that will be a square in our field.

3.6 Limitations of the Curve

If we try to use the above ideas for other elliptic curves, there are significant limitations on when this algorithm is actually practical. The main reason is because simple transformations can drastically increase the running time of the algorithm. Essentially, if we have to take a square root for every choice of y, rather than for all choices of y, then this algorithm is pointless. It would be faster - or similar in time - to simply choose x first, and calculate the square root of y^2 directly.

For instance, consider the elliptic curve:

$$x^3 + \alpha x^2 + b = y^2$$

First, we can turn this into the equation $x^3 + \alpha x^2 = u$. Then, naturally, we do a transformation $x \to 1/x$ (assuming $x \neq 0$) and multiply by x^3 to obtain the equation:

$$x^3 - \alpha x/u - 1/u = 0$$

We can apply to our algorithm to this curve. But u is different for each y, so we would be solving a square root for each y, rather than once for all y. This means we would probably be better off using the Standard Algorithm of choosing x and taking a square root to find y.

3.7 Summary of the Algorithm

We now present a summary of operations of the algorithm for the general case.

Algorithm 4

1. Begin with a normal basis in the field of 3^r elements where r is prime. The basis elements will be denoted a_0, a_1, $\cdots$, a_{r-1}. Let E be an elliptic curve

over $\mathbb{F}_{3^r}$ with the equation:

$$x^3 + \alpha x + b = y^2$$

2. Let $t = \alpha^{\frac{3^r+1}{4}}$.

3. Calculate t^2. It will either be α or $-\alpha$.

4. Now, transform this equation by $x \to t \cdot x$. We now have the following curve:

$$x^3 \pm x = (y^2 - b)/(\alpha t) \tag{3.2}$$

5. If α is a square, then we are in the positive case. That means we can use an error term. Let z be defined by $z_k = (-1)^k$ for all k.

6. Now we perform our major computation. Choose y. Now let $u = (y^2 - b)/(\alpha t)$. We now have the equation:

$$x^3 \pm x = u \tag{3.3}$$

7. Represent u by the normal basis. Let u_k be defined such that

$$u = a_0 u_0 + a_1 u_1 + ... + a_{r-1} u_{r-1}$$

8. Choose $x_0 \in \mathbb{F}_3$. Recursively find x_k by performing the computations:

$$x_{k+1} = u_k \mp x_k \quad \text{for } 0 \leq k \leq r-1$$

9. Now check if $x_0 = x_r$. If it is, then move to the step 10. If it isn't, then it depends on if α or $-\alpha$ is a square. If $-\alpha$ is a square, choose next y and go

back to step 6. If α is a square, then calculate $e = x_0 - x_r$. Let $x := x + ez$.

10. Transform $x \to x/t$ and we have (x, y) is a point on the elliptic curve.

This algorithm takes $O(r^2)$ operations. The computation that dominates is the transforming $x \to tx$ and reversing it. If α is a square, then it will always take the same amount of time as well.

4 Extending to Characteristic p

Algorithms 1-4 simply use the fact that cubing is linear in fields of characteristic three. Similarly, x^p is a linear operation in a field of characteristic p. So now we consider $\mathbb{F}_{p^r}$. We will discuss an algorithm for mapping into Hyperelliptic Curves in the following form:

$$x^p + \alpha x + b = y^2$$

4.1 The Base Cases

First, we look at the base cases. Consider the equation:

$$x^p + \beta x + b = y^2 \tag{4.1}$$

where $\beta \in \mathbb{F}_p$. We immediately see that the main part of our algorithm remains untouched. As β is in the base field, our computation remains in $\mathbb{F}_p$ rather than $\mathbb{F}_{p^r}$. If $\sum_{k=0}^{r-1} x_k a_k$ is the representation of x in the normal basis, then $\sum_{k=0}^{r-1} (x_k\beta)a_k$ is the representation of βx with the normal basis. Therefore we can immediately create an algorithm as follows:

Algorithm 5

1. Begin with a normal basis in the field of p^r elements where p and r are prime. The basis elements will be denoted a_0, a_1, $\cdots$, a_{r-1}. Let E be a

hyperelliptic curve over $\mathbb{F}_{p^r}$ with the equation:

$$x^p + \beta x + b = y^2$$

2. Choose y, and let $u = y^2 - b$ to obtain the equation $x^p + \beta x = u$.

3. Define u_k by the representation of u in the normal basis:

$$u = u_0 a_0 + u_1 a_1 + \cdots + u_{r-1} a_{r-1}$$

4. Let x_0 be a fixed element of $\mathbb{F}_p$ (for instance, let x_0 be 0). Let x_k be determined by the series of computations:

$$\begin{aligned} x_1 &= u_0 - \beta x_0 \\ x_2 &= u_1 - \beta x_1 \\ &\cdots \\ x_r &= u_{r-1} - \beta x_{r-1} \end{aligned}$$

5. Check that $x_r = x_0$. If it does, then (x, y) is a point on the hyperelliptic curve. If not, move on to the next step.

6. If $\beta = -1$, then choose the next y by some ordering and go back to Step 2. Otherwise, let $\gamma = -1/(1 + \beta^r) \mod p$, and $e = x_0 - x_r$. Define z by its coordinates in the normal basis:

$$z_k = (-1)^k \beta^k \gamma$$

7. Let $z = z_0a_0 + z_1a_1 + \cdots + z_{r-1}a_{r-1}$. Let $\hat{x} = x + ze$, and $(\hat{x}, y)$ is a point on the elliptic curve.

We see many similarities between this algorithm and the base cases of the characteristic 3 case. For instance, in Step 6. If $\beta = -1$, then we get a p-to-one map. As before, if our choice x_0 does not work, then none will work, and we simply have to change our choice of y. The speed of the algorithm becomes slower as p increases, because it's a one in p chance that we'll obtain a solution. If $\beta \neq -1$, then we have a one-to-one map. As before, we obtain an error vector, the calculation of which will be described in the next section.

After the initial setup (Step 1), which is relatively trivial, this algorithm takes $O(r^2)$ operations. The computation of u dominates the algorithm, which is the only part that takes $O(r^2)$ steps. If $\beta = -1$ then we have convergence based on the size of p.

4.2 Error Term Computation

Assume $\beta \neq -1$. Let $e = x_0 - x_r$. Let $\gamma = -(1/(1+\beta^r)) \mod p$. Then commence the operation $\hat{x_0} = x_0 + \gamma e$.

The transformation may look different, but it is actually the generalization of the one in characteristic three, because $\gamma = 1$ in $\mathbb{F}_3$ with $\beta = 1$.

Now we'll show why this works.

$$\begin{aligned}
\hat{x}_1 &= u_0 - \beta\hat{x}_0 \\
\hat{x}_1 &= u_0 - \beta(x_0 + \gamma e) \\
\hat{x}_1 &= x_1 - \beta\gamma e \\
\hat{x}_2 &= x_2 + \beta^2\gamma e \\
&\ldots \\
\hat{x}_k &= x_k + (-1)^k\beta^k\gamma e
\end{aligned}$$

We know r is odd as we assumed r is a large prime. So now we just have to check:

$$\begin{aligned}
\hat{x}_0 &\overset{?}{=} \hat{x}_r \\
x_0 + \gamma e &\overset{?}{=} x_r - \beta^r\gamma e \\
x_0 - x_r &\overset{?}{=} -\gamma e(1 + \beta^r) \\
1 &= -\gamma(1 + \beta^r)
\end{aligned}$$

Therefore this error term is the one we want. We can now describe the exact vector to add to the term. Let z be this error vector. Define:

$$z_k = (-1)^k\beta^k\gamma \tag{4.2}$$

So for straightforward calculation, one only needs to let $\hat{x} = x + ze$ by vector addition, and this point will be guarenteed to be on the elliptic curve. Therefore this algorithm is not only extremely fast but will always find a point in the same amount of time. Not only that, but γ does not depend on e. One only needs to calculate this γ once for each elliptic curve.

It should also be pointed out that, in general, p will be significantly smaller than r, so β^k will cycle around values over and over again.

4.3 The General Case

We now consider the general equation:

$$x^p + \alpha x + b = y^2$$

where $\alpha \in \mathbb{F}_{p^r}$. We will show later that there exist $\beta \in \mathbb{F}_p$ and $t \in \mathbb{F}_{p^r}$ such that $\alpha = t^{p-1}\beta$. With this, we perform the transformation $x \to tx$ to obtain:

$$\begin{aligned} t^p x^p + t\alpha x &= y^2 - b \\ t^p(x^p + \beta x) &= y^2 - b \\ x^p + \beta x) &= ((y^2 - b)\beta)/t^p \\ x^p + \beta x &= ((y^2 - b)\beta)/(t\alpha) \\ x^p + \beta x &= u \end{aligned}$$

So the method works exactly as it should. There is exactly one calculation of $\sqrt[p-1]{\alpha}$ and it is independent of the y chosen. After that, there is only straightforward calculation for each choice of y.

We will now find β and t.

4.4 Finding t and β

Lemma: Let $\gcd(r, p-1) = 1$ (which we assumed as r is a large prime) and let $k = -1/r \mod (p-1)$. If there exists t such that $t^{p-1} = \alpha$ then we can compute

t by the following:

$$t = \sqrt[p-1]{\alpha} = \alpha^{\frac{k \cdot p^r - k + p - 1}{(p-1)^2}}$$

Proof: First we need to show that the exponent is an integer so this operation is well defined:

$$\begin{aligned} kr &= -1 \mod p-1 \\ k(1+1+1+\cdots+1) &= -1 \mod p-1 \\ k(1+p+p^2+\cdots+p^r)+1 &= 0 \mod p-1 \\ k(p^r-1)+(p-1) &= 0 \mod (p-1)^2 \end{aligned}$$

Therefore the exponent is an integer and this is a well defined function. Now we show that this formula does in fact give a $(p-1)$st root of α, if $\alpha = t^{p-1}$ for some $t \in \mathbb{F}_{p^r}$.

$$\begin{aligned} &\alpha^{\frac{kp^r-k+p-1}{(p-1)^2} \cdot (p-1)} \\ &= \alpha^{\frac{kp^r-k+p-1}{(p-1)}} \\ &= \alpha^{\frac{k(p^r-1)}{(p-1)}} \cdot \alpha \end{aligned}$$

The first factor is a $(p-1)$st root of unity in $\mathbb{F}_{p^r}$. These are the elements in $\mathbb{F}_p$. That means that this root of unity is exactly β.

Therefore, look at t^{p-1}. Let $\beta = \alpha/t^{p-1}$. By definition, this gives us the element that we are looking for. We have now found β and t given any α such that $\alpha = t^{p-1}\beta$.

4.5 Limitations

Once again, this algorithm has its limitations. If the transformation of x is too complex, it will end up being a situation where one computes $p-1$ roots for each choice of y rather than once for all y. Take the polynomial:

$$x^p + x^{p-1} + b = y^b \tag{4.3}$$

Once again, choose y and reduce this to $x^p + x^{p-1} = u$. Now transform $x \to 1/t$ and multiply both sides by t^p to obtain:

$$ut^p - x = -1$$
$$t^p - x/u = -1/u$$

We can only solve this by taking a $p-1$ root of $-1/u$ for each choice of y, so the algorithm has no advantage over the Standard Algorithm.

4.6 Summary of the Algorithm

We now give a summary of the steps of the complete algorithm in characteristic p.

Algorithm 6

1. Begin with a normal basis in the field of p^r elements where p and r are prime. The basis elements will be denoted a_1, a_2, $\cdots$, a_{r-1}. Let E be an elliptic curve over $\mathbb{F}_{p^r}$ with the equation:

$$x^p + \alpha x + b = y^2$$

2. Let $k = -1/r \mod p-1$. Define t to be:

$$t = \alpha^{\frac{kp^r - k + p - 1}{(p-1)^2}} \tag{4.4}$$

3. Let $\beta = \alpha / t^{p-1}$.

4. If $\beta \neq -1$ then $x \to x^p + \beta x$ is one-to-one, and we can use the error vector. Let $\gamma = -1/(1+\beta^r) \mod p$, Define error vector z by:

$$z_k = (-1)^{k+1}\gamma$$

5. Perform the transformation $x \to tx$ which will turn our equation into:

$$x^p + \beta x = \frac{(y^2 - b)\beta}{\alpha t}$$

6. Choose y, and let $u = \frac{(y^2-b)\beta}{\alpha t}$, and put that in the normal basis.

7. Define u_k to be the representation of u in the normal basis:

$$u = u_0 a_0 + u_1 a_1 + ... + u_{r-1} a_{r-1}$$

8. Choose x_0, and define x_k by the computations:

$$x_{k+1} = u_k - \beta x_k \text{ for } k = 0, 1, \cdots, r-1$$

9. Check if $x_0 = x_r$. If it is then move to step 10. If not, check if $\beta = -1$. If $\beta = -1$, choose the next y and go back to step 7. If $\beta \neq -1$, then let $e = x_0 - x_r$, and then let $x := x + ez$.

10. Perform the transformation $x := x/t$ and (x, y) is a point on the elliptic

curve.

The initial setup of our algorithm takes $O(r^3)$ operations (Steps 1-4), due to exponentiation involved in finding β and t. However, after the initial setup, each point calculated on the curve takes only $O(r^2)$ operations. The transformation $x \to tx$ and back (Steps 5 and 10) dominates the rest of the algorithm as Step 8 only takes $O(r)$ operations.

4.7 Conclusions

We have extended our original algorithm to several different scenarios to obtain fast hashing into curves we previously could not. We avoided taking unnecessary square roots and $(p-1)$st roots so that the algorithm maintained its speed. This technique speeds up this specific part of the BLS Signature Scheme and any analagous signature schemes with hyperelliptic curves.

Bibliography

[1] Paulo S. L. M. Barreto and Hae Y. Kim, “Fast hashing onto pairing-friendly elliptic curves over ternary fields,” Revista de Engenharia de Computação e Sistemas Digitais 2, pp. 19–28.

[2] K. Hensel, Ueber die Darstellung der Zahlen eines Gattungsbereiches für einen beliebigen Primdivisor, J. Reine Angew. Math. 103 (1888), 230-237.

[3] von zur Gathen, Joachim; Giesbrecht, Mark “Constructing normal bases in finite fields,” J. Symbolic Comput. 10 (1990), no. 6, 547-570.