Improving the Efficiency of Elliptic Curve Scalar Multiplication using Binary Huff Curves

Gerwin Gsenger and Christian Hanser

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology (TUG), Inffeldgasse 16a, 8010 Graz, Austria
{gerwin.gsenger@student.tugraz.at|christian.hanser@iaik.tugraz.at}

Abstract. In 2010, Joye et. al brought the so-called Huff curve model, which was originally proposed in 1948 for the studies of diophantine equations, into the context of elliptic curve cryptography. Their initial work describes Huff curves over fields of large prime characteristic and details unified addition laws. Devigne and Joye subsequently extended the model to elliptic curves over binary fields and proposed fast differential addition formulas that are well-suited for use with the Montgomery ladder, which is a side-channel attack resistant scalar multiplication algorithm. Moreover, they showed that, in contrast to Huff curves over prime fields, it is possible to convert (almost) all binary Weierstrass curves into Huff form.
We have implemented generalized binary Huff curves in software using a differential Montgomery ladder and detail the implementation as well as the optimizations to it. We provide timings, which show speed-ups of up to 7.4% for binary NIST curves in Huff form compared to the reference implementation on Weierstrass curves. Furthermore, we present fast formulas for mapping between binary Weierstrass and generalized binary Huff curves and vice versa, where in the back conversion step an implicit y-coordinate recovery is performed. With these formulas, the implementation of the differential Montgomery ladder on Huff curves does not require more effort than its counterpart on Weierstrass curves. Thus, given the performance gains discussed in this paper, such an implementation is an interesting alternative to conventional implementations. Finally, we give a list of Huff curve parameters corresponding to the binary NIST curves specified in FIPS 186-3.

1 Introduction

In 1985, Neal Koblitz [10] and Victor S. Miller [14] both discovered independently that elliptic curves over finite fields can be used for public key cryptography. By now, elliptic curves have become an important concept within public key cryptography. This is mainly due to small key sizes compared to other public key cryptosystems, such as RSA, which lower the requirements for CPUs as well as the memory footprint. Elliptic curves have a group structure and their cryptographic security relies on the difficulty of solving the elliptic curve discrete logarithm problem (ECDLP). Traditionally, Weierstrass equations have been

used to describe elliptic curves. Nevertheless, over time, especially in the last couple of decades, new models to describe elliptic curves have been found, such as the Montgomery form, the Edwards form [2] and the like.

In 1948, Huff was studying a diophantine problem, for which he introduced a new elliptic curve model [8]. In 2010, Joye et. al [9], studied Huff's model over fields of odd characteristic and introduced formulas for fast point arithmetics. Each Huff curve defined over $\mathbb{F}_p$ contains a subgroup isomorphic to $\mathbb{Z}_4 \times \mathbb{Z}_2$ which, unfortunately, implies that there are no standardized prime curves that can be expressed in Huff form, as all these curves have cofactors equal to one. The Huff model has subsequently been extended by Devigne and Joye [5] to cover elliptic curves defined over binary fields $\mathbb{F}_{2^m}$. Furthermore, in [5], they present a unified addition law and give very fast differential addition formulas for binary Huff curves, which are well-suited for use with the Montgomery ladder, which is a side-channel attack resistant scalar multiplication algorithm.

For $m \geq 4$, every Weierstrass curve over $\mathbb{F}_{2^m}$ is birationally equivalent to a generalized Huff curve. Therefore, we are free to convert all standardized binary curves into corresponding (generalized) binary Huff curves. The authors of [5], give formulas for mapping points between binary Weierstrass curves and binary Huff curves, using an intermediate Weierstrass curve isomorphic to the original curve in short Weierstrass form.

1.1 Contribution

We have implemented generalized binary Huff curves in software, where for scalar multiplication we use a differential Montgomery ladder, which is a good choice for software implementations running in server environments, as it is resistant to remote timing attacks [4, 3, 18]. The differential Montgomery ladder is based on the differential addition formulas given by Devigne and Joye in [5]. The timings show that for the binary NIST curves, this implementation is up to 7.4% faster than the reference implementation of the differential Montgomery ladder on the original Weierstrass curves, to which we have applied the same optimizations.

The conversion between binary Weierstrass and generalized binary Huff curves requires two steps. As generalized binary Huff curves are birationally equivalent to a class of Weierstrass curves, which are themselves isomorphic to curves in short Weierstrass form, it is necessary to apply a birational equivalence as well as an isomorphism for the conversion. Additionally, after applying the differential Montgomery ladder on binary Huff curves and mapping the result back to the corresponding Weierstrass curve, a y-coordinate recovery is necessary. We address this issue by providing fast all-in-one, back-and-forth conversion formulas. These back-and-forth conversions turn out to be almost as efficient as the y-coordinate recovery on Weierstrass curves, which is required anyway after using the differential Montgomery ladder on Weierstrass curves, which preserves most of the savings obtained through the faster differential addition formulas. When using the formulas presented in this paper, the implementation of the differential Montgomery ladder on Huff curves does not require more effort than its counterpart on Weierstrass curves. Thus, given the performance gains discussed in

this paper such an implementation is an interesting alternative to conventional implementations.

Finally, we present curve parameters for Huff curves, which are birationally equivalent to the NIST-recommended binary Weierstrass curves specified in FIPS 186-3 [6].

1.2 Outline

In Section 2 and 3 we are going to talk about elliptic curves in general and the Huff curve model in particular. Section 4 gives the Huff curve parameters we have derived for the NIST-recommended curves. It details the implementation, the improvements we have achieved, shows the benchmark results and draws a comparison to the reference implementation. At last, Section 5 concludes the paper.

2 Preliminaries

An elliptic curve over the field $\mathbb{K}$ is a plane, smooth curve described by the Weierstrass equation:

$$E : Y^2 + a'_1XY + a'_3Y = X^3 + a'_2X^2 + a'_4X + a'_6, \tag{1}$$

where $a'_1, a'_2, a'_3, a'_4, a'_6 \in \mathbb{K}$. The set $E(\mathbb{K})$ of points $(x, y) \in \mathbb{K}^2$ satisfying Equation (1) plus the point at infinity $\mathcal{O} = (0 : 1 : 0)$, which is the neutral element, forms an additive Abelian group. The addition of points on an elliptic curve is achieved using the *chord-and-tangent* method [16].

For cryptographic purposes, we are interested in elliptic curves over finite fields $\mathbb{F}_q$, especially in curves over fields $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ with p being a large prime number. The security of elliptic curve cryptography is based on the assumption that in general there is no subexponential-time algorithm to solve the discrete logarithm problem in elliptic curve groups (ECDLP). Given $Q, P \in E(\mathbb{F}_q)$ the ECDLP constitutes the problem of finding a scalar $k \in \mathbb{Z}$ such that $Q = k \cdot P$.

In cryptography, usually non-supersingular curves are being used, as supersingular curves turned out to be susceptible to the MOV attack [13], which reduces the ECDLP to the DLP on finite fields, for which an efficient subexponential-time algorithm is known. Non-supersingular curves defined over $\mathbb{F}_{2^m}$, are described using the following short form of Equation (1):

$$Y^2 + XY = X^3 + a_2X^2 + a_6, \tag{2}$$

where $a_2, a_6 \in \mathbb{F}_{2^m}$ and $a_6 \neq 0$.

For binary field multiplications we are using the windowed left-to-right comb multiplier as detailed in [7]. This algorithm uses lookup-tables, which can be cached and reused to speed up multiplications with frequently used values, such as curve parameters. In the following, let $\mathbf{M}$ and $\mathbf{S}$ denote the costs of one field multiplication and one field squaring, respectively. Moreover, let $\mathbf{m_w}$ stand for the cost of the multiplication with some value, for which the lookup table of window size w has already been calculated.

2.1 Differential Addition and the Montgomery Ladder

Scalar multiplication is the most crucial and at the same time the most costly operation on elliptic curves and there are many ways to perform elliptic curve scalar multiplications. One way to do so is the so-called Montgomery ladder, which is both side-channel resistant and reasonably fast. Its side-channel resistance is due to the fact that at each ladder step, regardless of the currently processed bit of the scalar k, one point doubling operation as well as one point addition operation are performed and only the order of execution is changed. Thus, in order to mitigate side-channel attacks, the Montgomery ladder is widely used, especially in hardware implementations. In the context of software implementations, side-channel resistance is less important. Still, there are scenarios, such as cryptographic implementations running on servers, where remote timing attacks can be performed (cf. [4, 3, 18]) and, therefore, timing-attack proof implementations are required. This issue can again be addressed with the use of the Montgomery ladder, as its execution time is constant for fixed scalar lengths.

For the fast application of the Montgomery ladder, differential addition and doubling formulas are necessary. Luckily, these two operations can be performed very efficiently on Huff curves. By *differential point addition*, we mean the computation of $w(P+Q)$ from $w(P)$, $w(Q)$, and $w(P-Q)$, where $w(\cdot)$ is a coordinate function for which typically $w(P) = w(-P)$ is demanded. Note that the difference $w(P-Q)$ required for the differential addition steps within the Montgomery ladder is fixed, i.e., the difference of the resulting points stays invariant throughout the whole process. For differential addition on Weierstrass curves, the knowledge of y-coordinates is irrelevant, as the computation of x-coordinates does not involve them, which is reflected in the definition of $w(\cdot)$. Clearly, this saves computational effort, as the y-coordinate of intermediate results is not computed. Analogously, we mean by differential doubling the computation of $w(2P)$ from $w(P)$.

On Weierstrass curves, the fastest differential addition formulas are due to Bernstein and Lange (see [1]). From the results of Stam [17], they derive a representation, called XZ-coordinates, featuring fast differential addition and doubling formulas. Per bit of scalar k, these formulas require $\mathbf{5M + 1m_8 + 4S}$ for P known a priori and $\mathbf{6M + 2m_8 + 5S}$ otherwise.

The Montgomery ladder for projective differential addition formulas is shown in Algorithm 1. Here, the addition and the doubling steps are carried out using differential addition and differential doubling formulas, where in the former case the input is $W(2P)$, $W(P)$ and the invariant, consequently, is $W(P)$. Note that $W(\cdot)$ stands for the projective version of the coordinate function $w(\cdot)$.

After application of the Montgomery ladder, the y-coordinate of the product $k \cdot P = (x_1, y_1)$ can be restored efficiently and, hence, there is no need to compute it in the intermediate steps. Restoring the y-coordinate for affine coordinates, also means simultaneously scaling the x-coordinate, in order to avoid multiple

Algorithm 1 Differential Montgomery Ladder

Require: A point P on E and a scalar $k = (k_{l-1}, \ldots, k_0)_2 \in \mathbb{Z}$
Ensure: $W(k \cdot P) = (X_1 : Z_1)$
  $(X_1 : Z_1) = W(P)$ and $(X_2 : Z_2) = W(2P)$
  for $i = l - 2$ **downto** 0 **do**
    if $k_i = 0$ **then**
      $(X_1 : Z_1) = 2(X_1 : Z_1)$, and
      $(X_2 : Z_2) = (X_1 : Z_1) + (X_2 : Z_2)$
    else
      $(X_1 : Z_1) = (X_1 : Z_1) + (X_2 : Z_2)$, and
      $(X_2 : Z_2) = 2(X_2 : Z_2)$
    end if
  end for

inversions. It works in the following way, as detailed in [11] and in [17]:

$$x_1 = \frac{\bar{Z}\bar{X}Z_2X_1}{\bar{Z}\bar{X}Z_2Z_1} \quad \text{and} \quad y_1 = \bar{Y} + \frac{\gamma((X_1\bar{Z} + \bar{X}Z_1)\gamma + Z_1Z_2\bar{X}^2 + Z_1Z_2\bar{Z}\bar{Y})}{\bar{Z}\bar{X}Z_2Z_1},$$

where $\gamma = X_2\bar{Z} + \bar{X}Z_2$ and $P = (\bar{X} : \bar{Y} : \bar{Z}) \neq \mathcal{O}$. There are two special cases to check, see [11] for more details on that. Using these formulas requires **1I**+**10M**, when assuming $\bar{Z} = 1$, i.e., P to be scaled.

3 The Binary Huff Curve Model

This section is mainly a recapitulation of the research done by Joye et. al in [5] and [9], since we need these facts in Section 4 in order to derive compact formulas for mapping between binary Weierstrass and binary Huff curves.

As defined in [5], binary Huff curves are a new class of elliptic curves. For every Weierstrass curve defined over $\mathbb{F}_{2^m}$ with $m \geq 4$, there is a birationally equivalent generalized binary Huff curve. A *generalized binary Huff curve* is given by the set of projective points $(X : Y : Z)$ over $\mathbb{F}_{2^m}$ satisfying the subsequent equation:

$$E_H : aX(Y^2 + fYZ + Z^2) = bY(X^2 + fXZ + Z^2) \tag{3}$$

with $a, b, f \in \mathbb{F}_{2^m}^*$ and $a \neq b$. Three points at infinity, i.e., $(1 : 0 : 0), (0 : 1 : 0)$, and $(a : b : 0)$, satisfy Equation (3). The affine version of Equation (3) is birationally equivalent to the subsequent binary Weierstrass curve:

$$E_{W'} : v(v + (a + b)fu) = u(u + a^2)(u + b^2), \tag{4}$$

which itself is isomorphic to the short Weierstrass form through the following isomorphism:

$$\begin{aligned} \Theta : E_W(\mathbb{F}_{2^m}) &\longrightarrow E_{W'}(\mathbb{F}_{2^m}) \\ (u, v) &\longmapsto \left(\mu^2 u, \mu^3 \left(v + su + \sqrt{a_6}\right)\right), \end{aligned}$$

and its inverse:

$$\begin{aligned} \Omega : E_{W'}(\mathbb{F}_{2^m}) &\longrightarrow E_W(\mathbb{F}_{2^m}) \\ (u', v') &\longmapsto \left(\nu^2 u', \nu^3 v' + s\nu^2 u' + \sqrt{a_6}\right) \end{aligned}$$

with $\mu = (a+b)\, f$, $\nu = \mu^{-1}$, and $s \in \mathbb{F}_{2^m}$ satisfying $s^2 + s = a_2 + f^{-2}$. The birational equivalence

$$\begin{aligned} \Phi : E_{W'}\left(\mathbb{F}_{2^m}\right) &\longrightarrow E_H\left(\mathbb{F}_{2^m}\right), \\ (u, v) &\longmapsto \left(\frac{b\left(u + a^2\right)}{v}, \frac{a\left(u + b^2\right)}{v + (a+b)\, fu}\right), \end{aligned}$$

maps from the Weierstrass curve $E_{W'}$ onto the Huff curve E_H. Its inverse looks as follows:

$$\begin{aligned} \Psi : E_H\left(\mathbb{F}_{2^m}\right) &\longrightarrow E_{W'}\left(\mathbb{F}_{2^m}\right), \\ (x, y) &\longmapsto \left(\frac{ab}{xy}, \frac{ab\left(axy + b\right)}{x^2 y}\right) \end{aligned}$$

The neutral element is $(0, 0)$ and the group law is based on the chord-and-tangent rule. Note, however, that the birational equivalences are not line-preserving. The inverse of some affine point $P = (x_1, y_1) \in E_H$ is

$$-P = \left(y_1 \frac{(\alpha x_1 + 1)}{(\beta y_1 + 1)}, x_1 \frac{(\beta x_1 + 1)}{(\alpha x_1 + 1)}\right), \tag{5}$$

with $\alpha = \frac{a+b}{b \cdot f}$ and $\beta = \frac{a+b}{a \cdot f}$. The projective homogenization of Equation (5) can be evaluated using $\mathbf{4M + 4m_8}$. Furthermore, there are four exceptional cases, which have to be dealt with separately. We refer the reader to [5] for more details on that. Compared to binary Weierstrass curves, where point inversion costs at most one field addition and, hence, is negligible, this operation is very expensive on binary Huff curves.

Projective point doubling, and point addition take $\mathbf{6M + 2m_8 + 6S}$ and $\mathbf{15M + 3S}$, respectively. The affine doubling formula looks as follows:

$$2P = \begin{cases} (1:0:0) & \text{if } x_1 = 1, \\ (0:1:0) & \text{if } y_1 = 1, \\ (a:b:0) & \text{if } x_1 y_1 = 1, \text{ and} \\ \left(\frac{f(a+b)x_1^2(1+y_1)^2}{b(1+x_1)^2(1+x_1y_1)^2}, \frac{f(a+b)y_1^2(1+x_1)^2}{a(1+y_1)^2(1+x_1y_1)^2}\right) & \text{otherwise.} \end{cases} \tag{6}$$

where $P = (x_1, y_1) \in E_H$. Also note that for subgroups not including the points at infinity a complete addition law exists, i.e., it can also be used for point doublings.

3.1 Differential Addition

For Huff curves, the authors of [5] give differential addition formulas for use with the Montgomery ladder. To do so, they introduce the coordinate function $w(x, y) = xy$, which fulfills the condition $w(P) = w(-P)$. From Formula (6), they derive the following differential doubling formula:

$$w(2P) = \begin{cases} \gamma \cdot \frac{w_1^2}{(1+w_1)^4} & \text{if } w_1 \neq 1, \text{ and} \\ (1:0) & \text{otherwise.} \end{cases} \tag{7}$$

where $\gamma = \frac{(a+b)^2}{ab}$ and $w_1 = w(P)$. Using the birational equivalence Φ and the multiplicative differential addition formula from [17], the authors of [5] obtain the subsequent addition formula:

$$w(P+Q) = \begin{cases} \frac{(w_1+w_2)^2}{\bar{w}(1+w_1w_2)^2} & \text{if } w_1w_2 \neq 1, \text{ and} \\ (1:0) & \text{otherwise.} \end{cases} \tag{8}$$

where $P \neq Q$, $w_1 = w(P)$, $w_2 = w(Q)$, and $\bar{w} = w(P-Q)$.

Working with affine coordinates is less satisfying due to the costly inversion required in each step. Hence, Devigne and Joye introduce projective WZ-coordinates. A point $P = (\frac{X_1}{Z_1}, \frac{Y_1}{Z_1}) \in E_H$ in WZ-coordinates, is represented by the tuple

$$(W:Z) = \begin{cases} (\theta w(P) : \theta) = (\theta X_1 Y_1 : \theta Z_1^2) & \text{if } P \neq (a:b:0), \text{ and} \\ (\theta : 0) & \text{otherwise.} \end{cases}$$

for some $\theta \in \mathbb{F}_{2^m}^*$. Now, when given $P = (W_1 : Z_1)$ and $Q = (W_2 : Z_2)$, the projective differential addition formulas look as follows:

$$\begin{cases} W(2P) = \gamma \cdot (W_1 Z_1)^2 \\ Z(2P) = (W_1 + Z_1)^4 \end{cases} \quad \text{and} \quad \begin{cases} W(P+Q) = \bar{Z}\,(W_1Z_2 + W_2Z_1)^2 \\ Z(P+Q) = \bar{W}(W_1W_2 + Z_1Z_2)^2, \end{cases}$$

where $W(P-Q) = (\bar{W}, \bar{Z})$ are the WZ-coordinates of $P-Q$. Obviously, the first requires $\mathbf{1M + 1m_8 + 3S}$, whereas the latter requires $\mathbf{6M + 2S}$, which can be improved to $\mathbf{4M + 2S}$ by computing $W_1Z_2 + W_2Z_1$ as $(W_1 + Z_1)(W_2 + Z_2) + (W_1W_2 + Z_1Z_2)$ and scaling the coordinate $(\bar{W} : \bar{Z})$, since it remains constant throughout the whole Montgomery ladder process.

4 Contribution

In this section we are going to describe a Huff curve implementation and the optimizations we have applied to it. Furthermore, we contrast its performance with a reference implementation that is based on Weierstrass curves, to which we have applied the same set of optimizations. Then, we present timings of both

implementations, which show that for the binary NIST curves, the Huff curve implementation is up to 7.4% faster than the reference implementation.

We also present the curve parameters for the binary Huff curves corresponding to the binary NIST curves and give formulas for back-and-forth conversion between binary Huff curves and binary curves in short Weierstrass form.

4.1 Deriving Curve Parameters

Deriving curve parameters $a, b, f \in \mathbb{F}_{2^m}$ from a given binary Weierstrass curve with parameters $a_2, a_6 \in \mathbb{F}_{2^m}$, works as follows [5, Proof of Proposition 2]:

1. Select an arbitrary $f \in \mathbb{F}_{2^m}$ so that $Tr(f^{-1}) = Tr(a_2)$ and $Tr(f^8 a_6) = 0$,
2. find a solution $t \in \mathbb{F}_{2^m}$ to the equation $t^2 + \left(f^4\sqrt{a_6}\right)^{-1} t + 1 = 0$, and
3. determine $a, b \in \mathbb{F}_{2^m}$ such that $\sqrt{t} = ab^{-1}$,

where $Tr(\cdot)$ is the trace function $Tr : \mathbb{F}_{2^m} \to \mathbb{F}_2, x \mapsto \sum_{i=0}^{m-1} x^{2^i}$. The parameters we have obtained for the NIST-recommended curves [6] plus the value s needed for Ω and Θ are listed in Table 1 and in Table 2, respectively.

4.2 Mapping between Huff and Weierstrass Curves

Let $P_W = (u, v)$ be a point on $E_W(\mathbb{F}_{2^m})$, $P_{W'} = (u', v') = \Theta(P_W) \in E_{W'}(\mathbb{F}_{2^m})$ and $P = \Omega(P_{W'})$ be the corresponding point on $E_H(\mathbb{F}_{2^m})$.

Weierstrass to Huff Curve: From the birational equivalence $\Phi \circ \Theta$, we derive the following formula for directly mapping P_W to WZ-coordinates:

$$(W:Z) = \left(ab\left(\mu^2 u + a^2\right)\left(\mu^2 u + b^2\right) : \mu^6 \left(v + su + \sqrt{a_6}\right)\left(\left(v + su + \sqrt{a_6}\right) + u\right)\right)$$

Through decomposition we get:

- $A = \mu^2 u$,
- $B = (A + a^2)$, $C = (A + b^2)$,
- $D = v_1 + su + \sqrt{a_6}$, $E = D + u$,
- $W = ab \cdot B \cdot C$, and
- $Z = \mu^6 \cdot D \cdot E$.

The values a^2, b^2, and $\sqrt{a_6}$ as well as the multiplication lookup tables for the values μ^2, μ^6, s, and ab can be determined a priori. For this setting, we get a total of $\mathbf{2M + 4m_8}$ for obtaining the WZ-coordinates of P_W. Of course, if P_W is fixed, one can precompute its scaled WZ-coordinates.

Huff to Weierstrass Curves: In order to convert the result $W(k \cdot P) = (W_1 : Z_1)$, obtained from applying the differential Montgomery ladder on E_H, back to E_W, we need to evaluate the birational equivalence $\Omega \circ \Psi$. The naive approach would be to, firstly, compute the x-coordinate of the corresponding point on $E_{W'}$

Curve	a	b	f
B-163	0x1	0x253f3c45a6d779b47e63758c35336f0679b42f4c0	0x6
K-163	0x1	0x20000000000000000000000000000000000000033	0x6
B-233	0x1	0x115b7c737bec7a5cc19212911c2bd03cadb9a29ddf9b1dc64b\8b3550fb3	0x3
K-233	0x1	0x1e5ff5c884156aaebbd38370425882dff04f04ba05a7f40740\82385c149	0x2
B-283	0x1	0x6a263cdd28c309d3d3068046747abe51375b0d763dccc64868\251918d59c21842dd4fe1	0x3
K-283	0x1	0x7a0fa1ffdaf44208a4efb593c405714e0fbc4423dd0db57384\89cb583073c2cae153d0d	0x2
B-409	0x1	0x3f2918c0e689aca093d4cf5a389aeda96eb5cdcb930617991d\09111a3f91dc7283123ef8ab912744e193c34c9bd3cd532e17b7	0x9
K-409	0x1	0x846538361ed11b7c42b9e302169a3ea16009df82f80a155d56\39d78d4ba8dd02284110d6b3fbc05dda9c0ed1c0d6316c72d676	0x2
B-571	0x1	0x3c0904534c17c94a947b971ee5e6a3f3fb917dd3b57d7ad1f6\ea35ec2593bae024934b8efe08d2a5bb97c4286665408d50f80c\afc8dfbee0011c03e785fe39c94c977d5e3a7f065	0xf
K-571	0x1	0x6a28a2cf6fb77a9485f438a79f8832d86c465b689fd80b3d9c\4b1ef40380b5d92f85044e450336618a69b209eb37ecdd23da7b\f7ee9e0fc1e98248edb0dc92f3510027be50cd2bb	0x2

Table 1: Generalized Huff curve parameters for NIST-recommended binary curves (cf. [6]).

Curve	s
B-163	0x4058c6f9ae170f30f3ec9def6b2ddf2a28f0c0872
K-163	0x4058c6f9ae170f30f3ec9def6b2ddf2a28f0c0872
B-233	0xe7b4bcdb4af3163783507af91971d49927298e32e548d55b3a0b602a42
K-233	0xb1ce7164613a37fed984a32b18d265ada947ed207757d373d4e139835a
B-283	0x21b24c4336e195a894fc9021fabac4e6988ff780c29522af3261508be\fb321108eda3fa
K-283	0x387fd1da986a7fa48458bf27d26d9162d60c2f7f6e3da61f30d215c6b\193e73f223326e
B-409	0x106176448c66d8e7d0ddc074d76277a7ac8093ee53499d108099266d9\82c68dae5cb61d5054b30ecfce3c3beebc8cbecb904fd0
K-409	0x15dedff38eafec7e43a277eea795fbb1d52e6075a8bfe6a275be0dcba\f6b781f8c9d37e4f414e8de3634d946434d9b6a6d62e20
B-571	0x12007d9377488ff6122ccce941d1cef856279188c8e82a6696a918b2f\ccd78353385beb5e972f83d491d22db627117ab1580dabd23c6e8adeb99\d3bdbc95d6fb645833ba6b4f182
K-571	0x2780c6d786569591600518d211a5d6fbd900d9b44a1e4e65016d2331d\d243a6b31db129832a46326c7e3fd9b43f900ee58ed165e550a3cc3a41f\b88b001fa79f398351bb7c35dea

Table 2: Auxiliary parameter s for mapping between NIST-recommended binary Weierstrass curves (cf. [6]) and the corresponding binary Huff curves.

as $u_1 = \frac{ab}{w(k \cdot P)}$ then, secondly, to map the result from $E_{W'}$ to E_W, and, finally, to recover its y-coordinate v_1 using the formula from [11]:

$$v_1 = \frac{(u_1 + u)\left((u_1 + u)(u_2 + u) + u^2 + v\right)}{u} + v, \tag{9}$$

which requires the additional value $u_2 = \frac{ab}{w((k+1) \cdot P)}$, which can be computed using the point $W((k+1)P) = (W_2 : Z_2)$ from the last step of the Montgomery ladder. All in all, this would require several inversions, which can be saved by combining the formulas. To do so, one can replace $u_1 = \frac{ab}{w(k \cdot P)}$ with $u_1 = \frac{\delta \cdot Z_1}{W_1}$, where $w(k \cdot P) = \frac{W_1}{Z_1}$ as $W(k \cdot P) = (W_1 : Z_1)$, $\delta = \frac{ab}{\mu^2}$ and $\mu = (a + b)f$, and so implicitly apply Ω. By inserting it into Equation (9), one derives the following formula:

$$v_1 = \frac{(\delta Z_1 + uW_1)\left((\delta Z_1 + uW_1)(\delta Z_2 + uW_2) + \left(u^2 + v\right) W_1 W_2\right)}{uW_1^2 W_2} + v,$$

Now, one can compute

$$U_1 = \delta Z_1 u W_1 W_2 \quad \text{and} \quad V_1 = \beta\left(\beta(\delta Z_2 + uW_2) + \left(u^2 + v\right) W_1 W_2\right) + uvW_1^2 W_2$$

with $\beta = \delta Z_1 + uW_1$ and obtain $(u_1, v_1) = \left(\frac{U_1}{uW_1^2 W_2}, \frac{V_1}{uW_1^2 W_2}\right)$. Common subexpression elimination yields the following relations:

- $A = \delta Z_1$, $B = \delta Z_2$,
- $C = uW_1$, $D = uW_2$,
- $E = A + C$, $F = B + D$,
- $G = W_1 W_2$, $H = (u^2 + v)$,
- $I = (C \cdot G)^{-1}$, $J = A \cdot G$,
- $K = J \cdot I$, $L = E \cdot I$,
- $u_1 = uK$, and
- $v_1 = L \cdot (E \cdot F + H \cdot G) + v$.

During the computation one can save the lookup tables of the intermediate values I and G, replacing $\mathbf{4M}$ by $\mathbf{2M} + \mathbf{2m_4}$. Moreover, δ is constant and multiplications with it require $\mathbf{1m_8}$. Finally, we recycle the lookup table of u_1, exchanging $\mathbf{3M}$ for $\mathbf{1M} + \mathbf{2m_4}$. In case the point $P_W = (u, v)$ is fixed, one can precompute the lookup tables of u and H using $w = 8$. Thus, one can trade $\mathbf{1M} + \mathbf{3m_4} + \mathbf{1S}$ for $\mathbf{4m_8}$. To sum this up, the back-conversion including the y-coordinate recovery costs $\mathbf{1I} + \mathbf{6M} + \mathbf{5m_4} + \mathbf{2m_8} + \mathbf{1S}$ in general, and $\mathbf{1I} + \mathbf{5M} + \mathbf{2m_4} + \mathbf{6m_8}$ for P_W fixed.

Implications: If the point P_W is not fixed, in which case one needs both back and forth conversions including a y-coordinate recovery, one obtains total costs of at most $\mathbf{1I} + \mathbf{8M} + \mathbf{5m_4} + \mathbf{6m_8} + \mathbf{1S}$. Using XZ-coordinates on short Weierstrass curves, the formulas for restoring the y-coordinate turn out to be faster by only $\mathbf{2m_8}$ and, hence, this difference is negligible. Taking the conversion to

WZ-coordinates into account, one gets a small difference of $\mathbf{2M} + \mathbf{6m_8}$, which only becomes significant if P_W is not known a priori. This minimal overhead for the conversion pays off, as for the scalar multiplication using WZ-coordinates one has $\mathbf{4M} + \mathbf{2m_8} + \mathbf{5S}$ contrary to $\mathbf{5M} + \mathbf{1m_8} + \mathbf{4S}$ per bit of the scalar k when using XZ-coordinates.

4.3 Implementation Details and Optimizations

The implementation is written in Java and is based on the NIST recommended elliptic curves `B-163, B-233, B-283, B-409` and `B-571` [6]. We are using fast reductions for the underlying binary fields and `long`-arrays for values thereof, which are in polynomial representation. Furthermore, we perform squarings in linear time using table lookups [15] and multiplications using the windowed left-to-right comb multiplier. The latter works with precomputed multiplication lookup tables and is due to Lim and Lee (cf. [12, 7]). For ordinary binary field multiplications we use windows of size $w = 4$ and for multiplications with values known beforehand, such as curve parameters and combinations thereof, we use windows of size $w = 8$. Additionally, we cache these lookup tables for curve parameters and recurring intermediate values in the addition/doubling formulas resulting in a quite reasonable speedup. We point out that both implementations only restore the y-coordinates without scaling the result and, thus, return projective coordinates. The coordinate $W(2P - P) = W(P) = \left(\bar{W} : \bar{Z}\right)$ gets scaled and, thus, one can ignore $\bar{Z}$. Furthermore, $\bar{W}$ is either fixed because of P being the base point, or fixed throughout the whole ladder steps as the point difference stays the same. In both cases one can store the multiplication lookup tables for $w = 8$. Hence, the costs of the differential addition drop from $\mathbf{4M} + \mathbf{2S}$ to $\mathbf{3M} + \mathbf{1m_8} + \mathbf{2S}$. With differential doubling requiring $\mathbf{1M} + \mathbf{1m_8} + \mathbf{3S}$, one gets an overall of $\mathbf{4M} + \mathbf{2m_8} + \mathbf{5S}$ per bit of scalar k. In case P is not fixed, one has additional one-time costs for scaling $\left(\bar{W} : \bar{Z}\right)$, that is $\mathbf{1I} + \mathbf{1M}$, and one-time costs for assembling its lookup tables with window size $w = 8$.

Finally, we emphasize that the implementation of the differential Montgomery ladder on Huff curves is straight-forward and does not require more effort than its counterpart on Weierstrass curves, especially when the all-in-one formulas presented in this paper are being used.

Timings: All benchmarks were carried out on an Intel Core i5-2540M running Ubuntu Linux 12.10/amd64 and OpenJDK 7u15/amd64 in server mode. Table 3 shows the timings of one application of the Montgomery ladder on the test platform using both WZ and XZ-coordinates on the binary NIST curves, where in the former case we present the timings for both cases, i.e., for fixed P fixed and random P. The performance gains achieved through precomputations, which are possible in case of fixed P, are reflected in the timings. Furthermore, all timings include the recovery of the y-coordinate as well as in case of WZ-coordinates the back-and-forth conversion between Weierstrass and Huff curves.

In Table 4, one can see the relative costs of one squaring and one multiplication with a curve parameter compared to an ordinary field multiplication. This

Coordinates	B-163 [ms]	B-233 [ms]	B-283 [ms]	B-409 [ms]	B-571 [ms]
XZ	0.709	1.315	1.896	4.203	8.403
WZ	0.692	1.251	1.826	4.040	8.143
Speedup	**2.46%**	**5.12%**	**3.83%**	**4.03%**	**3.19%**
WZ (P fixed)	0.662	1.224	1.778	3.928	8.039
Speedup	**7.10%**	**7.43%**	**6.64%**	**7.00%**	**4.53%**

Table 3: Timings of the Montgomery ladder for WZ and XZ coordinates using binary NIST curves.

analysis shows that trading **1M** for $\mathbf{1m_8 + 1S}$ per ladder iteration, as done by using WZ instead of XZ-coordinates, saves up to **0.54M** per ladder step on the test platform.

	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{233}}$	$\mathbb{F}_{2^{283}}$	$\mathbb{F}_{2^{409}}$	$\mathbb{F}_{2^{571}}$
1S =	**0.094M**	**0.080M**	**0.077M**	**0.061M**	**0.055M**
$\mathbf{1m_8}$ =	**0.369M**	**0.411M**	**0.387M**	**0.418M**	**0.430M**

Table 4: Costs of squarings and multiplications with curve parameters in relation to ordinary multiplications.

5 Conclusions

We have implemented binary Huff curves in software. For scalar multiplication, we were using a differential Montgomery ladder, based on the differential addition formulas given by Devigne and Joye in [5]. The timings show that for the binary NIST curves, this implementation is up to 7.4% faster than the reference implementation of the differential Montgomery ladder on the corresponding Weierstrass curves, to which we have applied the same set of optimizations.

We have presented fast all-in-one back-and-forth conversion formulas, which implicitly include the recovery of the y-coordinate. These conversions turn out to be almost as efficient as the y-coordinate recovery on Weierstrass curves, which is required anyway after using the Montgomery ladder on Weierstrass curves. This preserves most of the savings obtained through the faster differential addition formulas. Furthermore, we have presented curve parameters for Huff curves, which are birationally equivalent to the NIST-recommended binary Weierstrass curves.

Finally, we emphasize that when using the formulas presented in this paper, the implementation of the differential Montgomery ladder on Huff curves does not require more effort than its counterpart on Weierstrass curves. Thus, given the performance gains discussed in this paper such an implementation is an interesting alternative to conventional implementations.

References

1. Bernstein, D.J., Lange, T.: Explicit-formulas database. `http://www.hyperelliptic.org/EFD/index.html`
2. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In: ASIACRYPT. pp. 29–50 (2007)
3. Brumley, B.B., Tuveri, N.: Remote timing attacks are still practical. In: ESORICS. pp. 355–371 (2011)
4. Brumley, D., Boneh, D.: Remote timing attacks are practical. Computer Networks 48(5), 701–716 (2005)
5. Devigne, J., Joye, M.: Binary Huff curves. In: Proceedings of the 11th international conference on Topics in cryptology: CT-RSA 2011. pp. 340–355. CT-RSA'11, Springer-Verlag, Berlin, Heidelberg (2011)
6. Gallagher, P., Furlani, C.: Fips pub 186-3 federal information processing standards publication digital signature standard (dss) (2009)
7. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2003)
8. Huff, G.B.: Diophantine problems in geometry and elliptic ternary forms. Duke Math. J. 15, 443–453 (1948)
9. Joye, M., Tibouchi, M., Vergnaud, D.: Huff's model for elliptic curves. In: ANTS. pp. 234–250 (2010)
10. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation 48(177), pp. 203–209 (1987)
11. López, J., Dahab, R.: Fast multiplication on elliptic curves over GF(2^m) without precomputation. In: Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1717, pp. 316–327. Springer (1999)
12. López, J., Dahab, R.: High-speed software multiplication in F_{2^m}. In: INDOCRYPT. pp. 203–212 (2000)
13. Menezes, A., Okamoto, T., Vanstone, S.: Reducing elliptic curve logarithms to logarithms in a finite field. In: Proceedings of the twenty-third annual ACM symposium on Theory of computing. pp. 80–89. STOC '91, ACM, New York, NY, USA (1991)
14. Miller, V.S.: Use of elliptic curves in cryptography. In: Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85. pp. 417–426. Springer-Verlag New York, Inc., New York, NY, USA (1986)
15. Schroeppel, R., Orman, H.K., O'Malley, S.W., Spatscheck, O.: Fast key exchange with elliptic curve systems. In: CRYPTO. pp. 43–56 (1995)
16. Silverman, J.: The Arithmetic of Elliptic Curves, Graduate Texts in Mathematics, vol. 106. Springer (1986)
17. Stam, M.: On Montgomery-like representations for elliptic curves over GF(2^k). In: Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography: Public Key Cryptography. pp. 240–253. PKC '03, Springer-Verlag, London, UK, UK (2003)
18. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-vm side channels and their use to extract private keys. In: ACM Conference on Computer and Communications Security. pp. 305–316 (2012)