Multi-Dimensional Montgomery Ladders for Elliptic Curves

Daniel R. L. Brown

June 12, 2006

Abstract

Montgomery's ladder algorithm for elliptic curve scalar multiplication uses only the x-coordinates of points. Avoiding calculation of the y-coordinates saves time for certain curves. Montgomery introduced his method to accelerate Lenstra's elliptic curve method for integer factoring. Bernstein extended Montgomery's ladder algorithm by computing integer combinations of two points, thus accelerating signature verification over certain curves. This paper modifies and extends Bernstein's algorithm to integer combinations of two or more points.

1 Introduction

Montgomery observed that, for some elliptic curves, the x-coordinate of the point $P + Q$, where $+$ is elliptic curve addition, could be calculated from the x-coordinate of the three points P, Q and $Q - P$.

Using this observation, Montgomery proposed that kG could be computed in such by computing a sequence of pairs of x-coorindates of two points $P = sG$ and $Q = (s + 1)G$, for appropriately selected values of s. This sequence has property that $Q - R = G$, so the difference of the points in known, and therefore $P + Q = (2s + 1)G$ can be computed with using a y-coordinate. The other element of the next pair is either $2sG = 2P$ or $2(s + 1)G = 2Q$, either of whose x-coordinate can be computed without y-coordinates.

For certain special kinds of elliptic curve, computing only with x-coordinates is faster than other efficient implementation methods. For prime NIST curves, this does not seem to be the case. Montgomery has defined a class of prime field curves for which not using the y-coordinate provides some savings. For non-Koblitz binary fields, there are y-free formulae that are comparable in cost with at least some other efficient implementations.

Suppose that one wants to compute $k_1G_1 + \cdots + k_dG_d$, using x-coordinates only, or at least mostly. Bernstein gives an algorithm for doing this when $d = 2$. At each step, a triple of x-coordinates is computed. The three points whose x-coordinates are computed at each stage have differences of the form $l_1G_1 + l_2G_2$ where $l_1, l_2 \in \{-1, 0, 1\}$. Therefore Montgomery's formula can be used, once the x-coordinates of $G_1 + G_2$ and $G_1 - G_2$ are found using convential addition with y-coordinates. This paper modifies and extends Bernstein's algorith to $d \geqslant 2$.

Potential applications of $d > 2$ include: batch ECC operations, accelerating Lenstra's ECM factoring algorithm, exploiting expanded ECC certificates (which contain pre-computed multiples of a party's public key), to incremental hashing based on elliptic curves, to accelerate verification of ECDSA signatures. Careful analysis is needed in each case to ascertain whether the algorithm presented outperforms existing efficient alternatives.

2 The Algorithm

Suppose that we wish to compute $a_1P_1 + \cdots + a_dP_d$ using a Montgomery ladder, where the a_i are m bit positive integers. We do the following.

1. Execute the precomputation phase:
   (a) Precompute and store, or otherwise make available, all the $(3^d - 1)/2$ x-coordinates of points $c_1P_1 + \cdots + c_dP_d$ where $c_i \in \{-1, 0, 1\}$, and the c_i are not all zero.

2. Execute the matrix phase, as follows.
   (a) Allocate $(m+1) \times d$ bit matrix $A = (A_{nk})$ and two $(d+1) \times d$ bit matrices $B = (B_{jk})$ and C_{jk}. Rows of A and B are indicated as A_j and B_j, respectively.
   (b) Allocate $(d+1) \times (m+1)$ integer matrices $F = (F_{j,n})$ and $G = (G_{j,n})$, with entry values ranging from 0 to $d+1$.
   (c) Allocate an $(m+1)$-wide array of $\{-1, 0, 1\}$-valued $(d+1) \times d$ matrices D^j.
   (d) Initialize A such that $a_k = A_{0,k} + 2A_{1,k} + 2^2A_{2,k} + \ldots$.
   (e) Initialize B as follows:
      i. Let h be the number odd entries in row A_0 (that is, the number of odd a_k.
      ii. Let $B_h = A_0$.
      iii. For each $j < h$, let B_j be obtained from B_{j+1} by subtracting one from a 1 valued entry (the choice of entry is arbitrary).
      iv. For each $j > h$, let B_j be obtained from B_{j-1} by adding one to a 0 valued entry (the choice of entry is arbitrary).
   (f) Let $n = 0$.
   (g) Let $D^n = B$.
   (h) Set $D^n = D^n \circ (-1)^{A_1}$, meaning to negate the columns of D in corresponding to positions A_1 with value 1.
   (i) Set the value matrix C and entries of F and G as follows:
      i. Set $j \leftarrow 0$.
      ii. Set $R \leftarrow A_0 + A_1 \bmod 2$.
      iii. Let h be the Hammning weight of R.
      iv. Let $C_h = R$.
      v. Let $F_{0,n} = G_{0,n} = h$.
      vi. Set $R \leftarrow B_{j+1} + C_{F_{j,n}}$.
      vii. Let h be the Hamming weight of R.
      viii. If $h < F_{j,n}$ then
         A. Set $F_{j+1,n} = h$,
         B. Set $G_{j+1,n} = G_{j,n}$,
         Else
         A. Set $R \leftarrow B_{j+1} + C_{G_{j,n}}$,
         B. Let h be the Hammning weight of R,

C. Set $F_{j+1,n} = F_{j,n}$,

D. Set $G_{j+1,n} = h$,

ix. Set $C_h = R$.

x. If $j < d - 1$, then set $j \leftarrow j + 1$ and go back to Step 2(i)vi.

(j) Set $B \leftarrow C$.

(k) Drop row A_0 from A, so that $A_j \leftarrow A_{j+1}$.

(l) Unless $n = m$, set $n \leftarrow n + 1$, and go back to Step 2g.

3. Execute the point addition phase:

(a) Initialize points $Q_0, \ldots, Q_d$, as follows:

i. $Q_j \leftarrow B_{j,1}P_1 + \cdots + B_{j,d}P_d$. These points are among the precomputed points. Note that $Q_0 = \infty$ and $Q_d = P_1 + \cdots + P_d$.

(b) Compute points $R_0, \ldots, R_d$ as follows :

i. $R_j \leftarrow Q_{F_{j,n}} + Q_{G_{j,n}}$.

ii. When making above the computation, the difference of $Q_{F_{j,n}} - Q_{G_{j,n}}$ is given by row j of matrix D^n, which means one look up the difference among the pre-computed points. Thus the y-coordinate is not needed to compute R_j from the Q points.

(c) Set $Q_j \leftarrow R_j$ for each j.

(d) If $n > 0$, then set $n \leftarrow n - 1$ and go back to Step 3b.

(e) Let h be the number of a_k that are odd.

(f) Output Q_h.

2.1 Example

Suppose that one wants to compute $10P_1 + 14P_2 + 9P_3 + 11P_4$ using the algorithm above.

The binary representations of the multiples $10, 14, 9, 11$ are $1010_2, 1110_2, 1001_2, 1011_2$, respectively. Therefore, we initiliaze a matrix A as

$$A = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{1}$$

where each column represnts a multipes with least significant bit at the top. A bottom row of all zeroes is appended for bookkeeping purposes. During the execution of the algorithm, the top row of A will be popped off. Equivalently, one can just move a pointer down a row, with the pointer starting at the top row.

The three ways that A is used in the algorithm are (a) to initialize the matrix B, which is done using the top row only, (b) to update matrix B, which is done using the modulo two sum of the first and second rows, and (c) to add determines the signs of the D matrices, which is done using the second row.

To initiliaze matrix B, first we take the top row of A, which has weight two. Therefore $B_2 = A_0$ (indexing the top row with 0). Rows B_1 and B_0 are obtained by replacing 1's with 0's and rows B_3 and B_4, by replacing 0's with 1's:

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{2}$$

We have thus used A to initialized B. The second use (b) of A uses the sum of consecutive rows, that is $A_0 + A_1$, $A_1 + A_2$, $A_2 + A_3$ and $A_3 + A_4$, which we compute below:

$$A_0 + A_1 = \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix} \tag{3}$$
$$A_1 + A_2 = \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix} \tag{4}$$
$$A_2 + A_3 = \begin{pmatrix} 1 & 0 & 1 & 1 \end{pmatrix} \tag{5}$$
$$A_3 + A_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \tag{6}$$
$$\tag{7}$$

These sums of rows are used to update B and in turn determine the integer entries in the arrays F and G.

We now update B, putting the results temporarily in another matrix C. By definition $A_0 + A_1$ must be row of C, and since it has weight 3, it will be row C_3. Also we get $F_{0,0} = G_{0,0} = 3$.

Now consider $C_3 + B_1 = (1\ 1\ 1\ 1)$, which has weight 4, so must be row C_4, and we get $F_{0,1} = 3$ and $G_{0,1} = 4$. Although not specified in the actual algorithm, we actually know that the entries of G cannot decrease and as 4 is the maximum, we will have $G_{0,j} = 4$ for $j \geqslant 2$. When G does not increase, then F decreases by one, so we get $F_{0,2} = 2$, $F_{0,3} = 1$ and $F_{0,4} = 0$. Also, we know that the value of F or G that differs from the previous iteration gives the index of the newly determined row of C, so rows $C_2 = B_2 + C_4$, and $C_1 = B_3 + C_4$ and $C_0 = B_4 + C_4$. Now B is updated to this new value of C:

$$B \leftarrow C = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 4 \\ 3 \\ 2 \\ 0 \\ 1 \end{pmatrix} \tag{8}$$

where the column of integers indicates the order in which the rows of C were computed. The matrices F and G have been partially computed as follows:

$$F = \begin{pmatrix} 3 & . & . & . \\ 3 & . & . & . \\ 2 & . & . & . \\ 1 & . & . & . \\ 0 & . & . & . \end{pmatrix} \qquad G = \begin{pmatrix} 3 & . & . & . \\ 4 & . & . & . \\ 4 & . & . & . \\ 4 & . & . & . \\ 4 & . & . & . \end{pmatrix} \tag{9}$$

Now we repeat this process, starting with $A_1 + A_2$ which has parity two. Matrix B gets updated to

$$B \leftarrow C = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 4 \\ 1 \\ 0 \\ 2 \\ 3 \end{pmatrix} \tag{10}$$

with the column again indicating the order of computing the rows of the updated B. Matrices F and G have been further filled out as follows:

$$F = \begin{pmatrix} 3 & 2 & . & . \\ 3 & 1 & . & . \\ 2 & 1 & . & . \\ 1 & 1 & . & . \\ 0 & 0 & . & . \end{pmatrix} \qquad G = \begin{pmatrix} 3 & 2 & . & . \\ 4 & 2 & . & . \\ 4 & 3 & . & . \\ 4 & 4 & . & . \\ 4 & 4 & . & . \end{pmatrix} \tag{11}$$

Though most of the work to obtain this updates have been left out, one can can see without any work that $C_2 = A_1 + A_2$. Working out the computation of the other rows one on own's is encouraged to appreciate what is involved. The reader may by now have noticed that we have been skipping the computation of the D matrices. We defer this because in this example, we will just look back at all the values B and adjust their signs using rows of A. Another update of B is given by:

$$B \leftarrow C = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 4 \\ 2 \\ 1 \\ 0 \\ 3 \end{pmatrix}. \tag{12}$$

Matrices F and G have been further filled out as follows:

$$F = \begin{pmatrix} 3 & 2 & 3 & . \\ 3 & 1 & 2 & . \\ 2 & 1 & 1 & . \\ 1 & 1 & 1 & . \\ 0 & 0 & 0 & . \end{pmatrix} \qquad G = \begin{pmatrix} 3 & 2 & 3 & . \\ 4 & 2 & 3 & . \\ 4 & 3 & 3 & . \\ 4 & 4 & 4 & . \\ 4 & 4 & 4 & . \end{pmatrix} \tag{13}$$

The final update is

$$B \leftarrow C = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 4 \\ 3 \\ 2 \\ 1 \\ 0 \end{pmatrix}. \tag{14}$$

Matrices F and G have been further finalized as:

$$F = \begin{pmatrix} 3 & 2 & 3 & 4 \\ 3 & 1 & 2 & 3 \\ 2 & 1 & 1 & 2 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad G = \begin{pmatrix} 3 & 2 & 3 & 4 \\ 4 & 2 & 3 & 4 \\ 4 & 3 & 3 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \end{pmatrix}. \tag{15}$$

The difference matrices that we need are

$$D^0 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ 0 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 \end{pmatrix}, D^1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ 1 & -1 & 1 & 1 \end{pmatrix}, D^2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & -1 \end{pmatrix}, D^3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \tag{16}$$

which are obtained by taking the successive B matrices (excluding the last one), and negating the columns where the corresponding rows of A (excluding the top row) have a one.

The final bit matrix B, the integer matrices F and G and the difference matrices D^0, D^1, D^2, D^3 make up the plan used for the point addition phase. The matrix B says how to initialize the the sequence of points. The matrices F and G say how to update the sequence of points, by virtue of which elements to add. The matrices D^j say how to determine the differences of the points being added, which is needed to the x-only Montgomery addition laws.

We summarize the entire point addition phase in the following array

$$\begin{array}{cccc|cccc|cccc|cccc|cccc} 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 & 4 & 2 & 2 & 6 & 8 & 4 & 6 & 10 & 14 & 10 & 12 \\ 0 & 1 & 0 & 0 & 2 & 2 & 1 & 2 & 2 & 4 & 2 & 3 & 5 & 8 & 4 & 6 & 10 & 14 & 10 & 11 \\ 0 & 1 & 0 & 1 & 1 & 2 & 1 & 2 & 3 & 4 & 2 & 3 & 5 & 7 & 4 & 6 & 10 & 14 & 9 & 11 \\ 1 & 1 & 0 & 1 & 1 & 2 & 1 & 1 & 3 & 3 & 2 & 3 & 5 & 7 & 5 & 6 & 10 & 15 & 9 & 11 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 3 & 3 & 3 & 3 & 5 & 7 & 5 & 5 & 11 & 15 & 9 & 11 \end{array} \tag{17}$$

Think of this as five 5×4 matrices whose rows of the form (r_1, r_2, r_3, r_4) represent computation of the point $r_1 P_1 + \cdots + r_4 P_4$. It remains to explain how the plan leads to this computatain.

The first of the five matrices is the final value of bit matrix B. To obtain the subsequent matrices we use the previous matrix and the matrices F and G and the D^j matrices. However, we must use the columns of F and G and the matrices D^j in the *reverse* order that they were obtained.

For example, to obtain the third row of the second matrix, we note that that third row of the last column of F and G have the values two and four respecitvely, so therefore we add rows 2 and 4 of the first matrix to $(0, 1, 0, 1) + (1, 1, 1, 1) = (1, 2, 1, 2)$. Here we have added using integer vector arithmetic, we are no longer working modulo two. In fact, there is no big integer arithmetic in the point addition phase, just point addition, so we actually be doing a single point addition, as follows:

$$(P_2 + P_4) + (P_1 + P_2 + P_3 + P_4) = (P_1 + 2P_2 + P_3 + 2P_4) \tag{18}$$

where the additions inside the parenthesis on the left hand side were already done as part of the initialization, and the additions in the right hand side are achieve implicitly by the additions on the left hand side. The only point addition actually being done in this step is the one in the left hand side outside of the parentheses. Now $(1, 1, 1, 1) - (0, 1, 0, 1) = (1, 0, 1, 0)$, which is row 2 (counting with 0 from top as usual) of the last difference matrix.

2.2 Straightforward Simplifications

Evidently from the example, and from the theory numerous simplifications to the algorithm are possible. For example, the top row of B is always all zeros and the bottom row is always all ones. Similarly, the bottom row F is all zeros and the bottom row of G is always all d's. The top rows of F and G are identical and are determined by the weights of the sums of successive rows of A.

Because the columns of F weakly fall, and the columns of G weakly rise, with exactly one rise or fall between rows, it follows that we use single bits to indicate whether F falls or G rises. Thus integers can be used for only for the common top row of F and G, if this has some advantage.

2.3 Room for Improvement

It would be more convenient if the D matrices and columns of F and G could somehow be computed in the opposite order, because then one could interleave the bit matrix phase and the point addition phase. Technically, the total computation time might be the same whether or not one interleaves the two phases, but in practice interleaving may help slightly because mainly some implementations can do pipelining, effectively allowing independent computations to be done simultaneously. If the bit matrix phase has to be done first, during that the time the pipeline is unavailable for doing point additions.

3 Why It Works

This section describes why the algorithm works. To do this, several definitions, lemmas and theorems about matrices are introduced.

The algorithm has two phases, the matrix phase and the point addition phase. In the matrix phase, a plan, or ladder, is made of which intermediate integer combinations of the points that shall be computed on the way to the compute the target combination. The matrix phase uses integer matrices and does not involve any elliptic curve arithmetic. The point addition phase then uses this plan, or climbs the ladder, to add points together, for the most part not using y-coordinates, ultimately arriving at the target integer combination of points.

3.1 The Matrix Phase

We will use the following convention for indexing matrices. A matrix with d columns (rows) will have columns (rows) indexed by $1, 2, \ldots, d$, while a matrix with $d+1$ rows (columns) will have rows (columns) indexed by $0, 1, 2, \ldots, d$. The parameter d is called the *dimension*, and will be the number of points that we wish to combine in the application of the algorithm. Write M_j for the row of a matrix M indexed by j under the conventions stated above . Write e_i for an elementary row vector that has a one in position i and zero in all other positions.

Definition 1. *A state matrix is a $(d+1) \times d$ integer matrix S such that*

- *$S_j - S_{j+1} = \pm e_i$ for some i.*
- *S_j has j odd entries.*

An example state matrix is

$$S = \begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix} = \begin{pmatrix} 28 & 30 & 18 \\ 28 & 29 & 18 \\ 29 & 29 & 18 \\ 29 & 29 & 19 \end{pmatrix}. \tag{19}$$

Lemma 1. *Any integer row vector R is the row of some state matrix S.*

Proof. Suppose that R has j odd entries. Then set $S_j = R$. For $1 \leqslant i \leqslant j$, choose S_{i-1} be adding ± 1 to one of the odd entries of S_i. For $j \leqslant i < d$, choose row S_{i+1} by adding ± 1 to one of the even entries. □

Lemma 2. *If S is a state and $i < j$, the set of indices of odd entries of S_i is a subset of the corresponding set for row S_j.*

Proof. If suffices to prove this for $j = i + 1$, for which it is obvious. □

Definition 2. *A transition matrix is an $(d + 1) \times (d + 1)$ integer matrix M such that*

- *$M_j = e_i + e_{i+j}$ for some i.*
- *$M_j - M_{j+1} = \pm(e_i - e_{i+1})$ for some i.*

An example transition matrix is

$$M = \begin{pmatrix} M_0 \\ M_1 \\ M_2 \\ M_3 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 2 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}. \tag{20}$$

Theorem 3. *If M is a transition matrix and S is a state, then $T = MS$ is a state.*

Proof. Now $T_j = M_j S$. Since $M_j = e_i + e_{i+j}$, we have $T_j = S_i + S_{i+j}$. By Lemma 2, the odd entries of S_i will cancel i odd entries in S_{i+j}, leaving j entries as desired. To establish the second state property for T, we calculate that $T_j - T_{j+1} = (M_j - M_{j+1})S = \pm(e_i - e_{i+1})S = \pm(S_i - S_{i+1}) = \pm e_k$ for some k. □

Theorem 4. *If T is a state matrix, then there exists a unique state matrix S and unique transition matrix M, such that $T = MS$.*

Proof. We determine the rows of M in order, $M_0, M_1, \ldots, M_d$, and the rows of S in an order to be determined. Suppose that $\frac{1}{2}T_0$ has h odd entries. This implies that

$$M_0 = 2e_h \qquad \text{and} \qquad S_h = \frac{1}{2}T_0. \tag{21}$$

This is the base of the induction for determining the remaining rows. The induction will be on f and g such that $0 \leqslant f \leqslant g \leqslant d$, starting with $f = g = h$. At each stage of induction, rows $S_f, S_{f+1}, \ldots, S_g$ and $M_0, \ldots, M_{g-f}$ will have have been determined. Moreover we will have

$$M_{g-f} = e_f + e_g. \tag{22}$$

Let $j = g - f + 1$. By definition of transition matrices, we must have

$$M_j \in \{e_{f-1} + e_g, e_f + e_{g+1}\}. \tag{23}$$

In order for $T = MS$ to hold, this must imply that

$$T_j \in \{S_{f-1} + S_g, S_f + S_{g+1}\} \tag{24}$$

Therefore we need one of the following two equations to hold:

$$\begin{aligned} S_{f-1} &= T_j - S_g \\ S_{g+1} &= T_j - S_f \end{aligned} \tag{25}$$

Whichever these equations is valid will determine another row of the matrix S, namely, either S_{f-1} or S_{g+1}. If $T_j - S_g$ has $f-1$ odd entries then we must have the equation for S_{f-1}. If $T_j - S_f$ has $g+1$ odd entries, we must have the equation for S_{g+1}.

To show that exactly one of the conditions in (25) holds, note that $T_{j-1} = S_f + S_g$, and that T_j has one more odd entry than T_{j-1}. The extra odd entry in T_j can must in a position where S_f has an even entry. The entry in that position can be even or odd for S_g. If it is even for S_g, then the equation for S_{g+1} holds, and if odd, the equation for S_{f-1} holds. □

To illustrate, let us apply the algorithm in the proof to

$$T = \begin{pmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \end{pmatrix} = \begin{pmatrix} 24 & 28 & 18 & 22 \\ 24 & 28 & 18 & 23 \\ 24 & 29 & 18 & 23 \\ 25 & 29 & 18 & 23 \\ 25 & 29 & 19 & 23 \end{pmatrix}. \tag{26}$$

As in the proof, we compute the rows of M and S one at a time.

1. Row $T_0 = (24, 28, 18, 22)$, so $S_h = \frac{1}{2}T_0 = (12, 14, 9, 11)$. The number of odd entries of S_h is two, so $h = 2$, and $M_0 = 2e_2$.

2. The next row of S to compute must be $T_1 - S_2 = (12, 14, 9, 12)$, which has just one entry, so must be S_1. Thus $M_1 = e_1 + e_2$.

3. The next row of S to compute is either $S_0 = T_2 - S_2$ or $S_3 = T_2 - S_1$, as in (25). Because $T_2 - S_1 = (12, 15, 9, 11)$ has three odd entries, S_3 is the choice. Thus $M_2 = e_1 + e_3$.

4. The next row of S to comptue is either $S_0 = T_3 - S_3$ or $S_4 = T_3 - S_1$, depending on parity. The choice is $S_4 = T_3 - S_1 = (13, 15, 9, 11)$. Thus $M_3 = e_1 + e_4$.

5. The last of S to be determined is thus $S_0 = T_4 - S_4 = (12, 14, 10, 12)$, and as always $M_4 = e_0 + e_4$.

Therefore

$$\begin{pmatrix} 24 & 28 & 18 & 22 \\ 24 & 28 & 18 & 23 \\ 24 & 29 & 18 & 23 \\ 25 & 29 & 18 & 23 \\ 25 & 29 & 19 & 23 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 12 & 14 & 10 & 12 \\ 12 & 14 & 9 & 12 \\ 12 & 14 & 9 & 11 \\ 12 & 15 & 9 & 11 \\ 13 & 15 & 9 & 11 \end{pmatrix}. \tag{27}$$

Definition 3. *For any matrix A, define $|A|$ to the be maximum of the absolute values of the entries of A.*

Theorem 5. *If S and T are state matrices, M is a transition matrix, and $T = MS$, then $2|S| - 1 \leqslant |T| \leqslant 2|S|$.*

Proof. By definition, $|T| = |T_{j,k}|$ for some j and k. Now $T_{j,k} = S_{i,k} + S_{(i+j),k}$, by the properties of M. Therefore $|T| \leqslant |S_{i,k}| + |S_{(i+j),k}| \leqslant 2|S|$.

By definition, $|S| = |S_{i,k}|$ for some j and k. It is easy to see that there will some j such that $M_j = e_f + e_g$ where $i \in \{f, g\}$. Then $T_{j,k} = S_{f,k} + S_{g,k}$. Without loss of generality, suppose that $i = f$. By properties of state matrices, $S_{g,k} = S_{f,k} + \sigma$, with $\sigma \in \{-1, 0, 1\}$. But $|S_{g,k}| \leqslant |S_{f,k}| = |S|$, so $\sigma = 0$, or it has the opposite sign to $S_{f,k}$. Noting that $T_{j,k} = 2S_{i,k} + \sigma$, we see that $|T_{j,k}| \geqslant 2|S_{i,k}| - 1$, which gives $|T| \geqslant 2|S| - 1$. □

Corollary 6. *If S and T are state matrices, M is a transition matrix, and $T = MS$, then either $|S| < |T|$ or $|T| = 1$.*

Proof. Suppose that $|T| \leqslant |S|$. Then $2|T| - 1 \leqslant 2|S| - 1 \leqslant |T|$, and adding $1 - |T|$ to both ends gives $|T| \leqslant 1$. The case $|T| = 0$ is impossible, because T has odd entries. □

Lemma 7. *If S and T are state matrices, M is a transition matrix, $T = MS$, and $|T| = 1$, then $S = T$ and $M_j = e_0 + e_j$ for all j.*

Proof. Since $|T| = 1$, row $T_0 = 0$, because T_0 has all even entries with absolute values at most one. Suppose that M satisfies $M_j = e_0 + e_j$ and $S = T$. Then $S_0 = T_0 = 0$ and $T_j = S_0 + S_j = S_j$. Therefore $T = MS$. Because T factors uniquely, these choices of M and S are the only possibilities. □

Theorem 8. *Let S be a state matrix. Then S factors uniquely as a product*

$$S = AB \dots CT \tag{28}$$

where: $A, B, \dots, C$ are transition matrices, $C_0 \neq 2e_0$, and T is a state matrix with $|T| = 1$. Conversely, every such product is a state matrix.

Proof. Immediate from the previous results. □

To illustrate, we give the following factorization

$$\begin{pmatrix} 12 & 14 & 10 & 12 \\ 12 & 14 & 9 & 12 \\ 12 & 14 & 9 & 11 \\ 12 & 15 & 9 & 11 \\ 13 & 15 & 9 & 11 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$
$$\times \begin{pmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{29}$$

Definition 4. *Let A be a matrix or a vector. We write $A \geqslant 0$ and say that A is nonnegative, if all entries of A are nonnegative.*

Theorem 9. *If S and T are state matrices, M is a transition matrix, and $T = MS$, and $T \geqslant 0$ if and only if $S \geqslant 0$.*

Proof. Suppose that $S \geqslant 0$. Since $M \geqslant 0$ and the product of nonnegative matrices is nonnegative, we then have $T = MS \geqslant 0$.

Suppose that $S \ngeqslant 0$. Then S has a negative entry $S_{i,k} < 0$. It follows that $S_{l,k} \leqslant 0$ for all l. There exists a j such that $T_{j,k} = S_{f,k} + S_{g,k}$ for some f and g, with $i \in \{f, g\}$. It then follows that $T_{j,k} < 0$, and thus $T \ngeqslant 0$. □

Lemma 10. *If T is a state matrix, then it has rank d.*

Proof. Induction on $|T|$. For $T = 1$, it is obvious. Otherwise $T = MS$ for some state matrix S with $|S| < |T|$. Because M is invertible, matrices T and S have the same rank. □

Doubtlessly, because of the general interest of matrices in mathematics, there exist many different matrix factorization algorithms, perhaps some quite general. One may thus wonder if any of such algorithms cover the result above: here we speculate that they do not, mainly on the grounds that nobody would have had reason to consider the peculiar form of matrices that we needed to consider.

Incidentally, transition matrices have determinant ± 2, because only one permuation is embeddable into the nonzero entries of M. Therefore, transition matrices are always invertible. State matrices are not square, so have neither determinant nor inverse. Although not needed for the multi-dimensional Montgomery ladder algorithm, one can amend the definition of state matrices by preprend a zeroth column, all of whose entries are equal to a power of two. If $|T| = 1$, the power of two should be one, and otherwise it should be 2^t where t is the number of transition matrices in the (28). These amended state matrices are invertible and have determinant $\pm 2^t$.

The number of transition matrices is 2^d, which can be seen by starting from the last row, which is always $e_0 + e_d$, and noting that each successive row can have either the left entry or the right entry shifted one step in the opposite direction. The number of state matrices T with $|T| = 1$ is $2^d d!$, or just $d!$ if T is also nonnegative. Therefore, the number of state matrices that factor into the form in (28) with t transition matrices is $2^{dt}(2^d - 1)d!$, or just $2^{d(t-1)}(2^d - 1)d!$ when counting only nonnegative state matrices.

3.2 Difference Matrices

The point addition phases needs not only the transition matrices and the minimal state matrix, but it also needs the difference matrices (to be defined), so that Montgomery formula for point addition with x-coordinate only can be used.

Let M be a state matrix, such that it rows of the form $M_j = e_f + e_g$ with $f \leqslant g$. Define $\hat{M}$ such that it rows of the form $\hat{M}_j = e_f - e_g$. As usual, suppose that T is a state matrix factoring uniquely into as $T = MS$ where M is a transition matrix and S is a state matrix. The difference matrix corresponding to T is the matrix $D = \hat{M}S$. We note that a difference matrix D is actually a state matrix and that $|D| = 1$.

A simple rule may be used to determine the difference matrix $D = (d_{j,k})$ corresponding to state matrix $T = (t_{j,k})$. If $t_{j,k}$ is even, then $d_{j,k} = 0$. Otherwise $t_{j,k} \equiv d_{j,k} \bmod 4$.

3.3 Simplified Matrix Phase

One disadvantage of the methods given by §3.1 is the big integer arithmetic in the computation of the intermediate state matrices. A second disadvantage is the computation of the transition matrices in the matrix phase is in an order opposite to how they applied in the point addition phase, which means that matrix phase has to be complete before the addition phase can be begun. Therefore we develop some more theory that can be used to overcome these disadvantages.

Definition 5. *Fix some $1 \leqslant c \leqslant d$. Let S be any state matrix S. Let S' the matrix obtained by deleting column c and the deleting the row in which the entry of column c does not equal the value in the row below.*

Lemma 11. *If S is a state matrix for dimension d, then S' is a state matrix for dimension $d-1$.*

Proof. For any row vector R, let R' indicate the row vector with entry c deleted. For each j, we have $S'_j = (S_{j'})'$ where $j' \in \{j, j+1\}$, with the choice depending on whether j lies below the deleted row of matrix S. Now S'_j has j odd entries if $j' = j$, because the deleted entry of $S_{j'}$ is even, and S'_j has j odd entries if $j' = j+1$ because $S_{j'}$ has $j' = j+1$ odd entries and one odd entry, in position c, is deleted to give j odd entries.

Now $S'_j - S'_{j+1} \in \{(S_j - S_{j+1})', (S_j - S_{j+2})', (S_{j+1} - S_{j+2})'\}$, depending on whether j lies above, at, or below the deleted row of the matrix S. Because S is a state matrix will have $i \neq c$ such that $S_j - S_{j+1} = \pm e_i$, and $S_j - S_{j+2} = \pm e_i \pm e_c$ and $S_{j+1} - S_{j+2} = \pm e_i$. Upon deleteion of entry c, wee see that $S'_j - S'_{j+1}$ has the form $\pm e_i$. □

Theorem 12. *If S and T are state matrices, M is a transition matrix, and $T = MS$, then $T' = M'S'$ for some transition matrix M'.*

Proof. For any row vector R, let R' indicate the row vector with entry c deleted. For each j we have $T_j = S_f + S_g$ for some f and g. Therefore $(T_j)' = (S_f)' + (S_g)'$. We write $(T_j)' = T'_{j'}$ where $j' \in \{j, j+1\}$, with the choice of value depending on whether j lies above or below the row deleted from T. Similarly, we write $(S_k)' = S'_{k'}$ where $k' \in \{k, k+1\}$. Therefore, we have $T'_{j'} = S'_{f'} + S'_{g'}$. This means that $T' = M'S'$ for matrix M' such that $M'_{j'} = e_{f'} + e_{g'}$ with the same values of indices as used previously. We need only now to show that M' is a transition matrix.

To prove that M' is a transition matrix, we need to show both (a) that $g' - f' = j'$ and (b) that successive rows of M' differ by $\pm(e_i - e_{i+1})$ for some i.

To show (a), consider the parity of the entries of rows of state matrices S' and T' (in other words, look at the Hamming weight modulo two). Since $T'_{j'}$ has j' odd entries, the f' odd entries of row $S'_{f'}$ are subset of the g' odd entries of row $S'_{g'}$, we must have $j' = g' - f'$.

To show (b), consider that successive rows of T' differ in just one entry, with a difference of ± 1. Suppose that $T'_j - T'_{j+1} = \pm e_i$. We may also suppose, because the work above, that $T'_j = S_f + S_g$ where $g - f = j$ and $T'_{j+1} = S_e + S_h$ where $h - e = j+1$. Therefore $\pm e_i = S'_f + S'_g - S'_e - S'_h$. The columns of S' may be ordered according to in which row they become odd. The odd entried positions of $S'_f + S'_g$ correspond an interval, a contigous set of consecutive entries, in this ordering of the colmunns of S'. The same holds for $S'_e + S'_h$. The difference of these two sums of rows, has only one odd entry, so therefore, the corresponding intervals must overlap in all but one point. Moreover the interval for $S'_e + S'_h$ must extend that of $S'_f + S'_g$ by appending the next element either above or below. This implies that $(e, h) \in \{(f-1, g), (f, g+1)\}$, which implies condition (b) for M' be a transition matrix. □

An application of Theorem 12, is that the all integers appearing the successive state matrices need not be computed multiple times for occurrence. Instead, one can reduce each column to dimension $d = 1$ for computing the values of the entries. Dimension $d = 1$ is the classical Montgomery method. An advantage of this observation is that values in the intermediate state matrix entries may be computed easily from the bit representations of the initial state matrix, as the following theorem illustrates.

Theorem 13. *For $d = 1$, let T be a state matrix of the form*

$$T = \begin{pmatrix} 2a_0 + 2a_{s+1} + 4a_{s+2} + 2^3 a_3 + \dots \\ 1 + 2a_{s+1} + 4a_{s+2} + 2^3 a_3 + \dots \end{pmatrix} \tag{30}$$

where $a_i \in \{0, 1\}$. Then for $s \geqslant 1$, we have

$$T = M_{a_0+a_1} M_{a_1+a_2} \dots M_{a_{s-1}+a_s} \begin{pmatrix} 2a_s + 2a_{s+1} + 4a_{s+2} + \dots \\ 1 + 2a_{s+1} + 4a_{s+2} + \dots \end{pmatrix} \tag{31}$$

where $M_0 = M_2 = \left(\begin{smallmatrix} 2 & 0 \\ 1 & 1 \end{smallmatrix}\right)$ and $M_1 = \left(\begin{smallmatrix} 0 & 2 \\ 1 & 1 \end{smallmatrix}\right)$.

Proof. Induction on s. For $s = 1$, we can verify the result by inspecting each of four cases $(a_0, a_1) \in \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ individually. This covers the base of induction. Suppose (31) holds, then we shall show that it also holds with s replaced by $s + 1$. Let

$$S = \begin{pmatrix} 2a_s + 2a_{s+1} + 4a_{s+2} + \dots \\ 1 + 2a_{s+1} + 4a_{s+2} + \dots \end{pmatrix} \tag{32}$$

so that (31) becomes $T = M_{a_1} M_{a_1+a_2} \dots M_{a_{s-1}+a_s} S$. Now apply the base case of induction to matrix S, getting $S = M_{a_s+a_{s+1}} U$, where U has the desired form. $\square$

This result represents a simplification to the procedure in this $d = 1$, by overcoming both of the disadvantages mentioned earlier. The transition matrices can be computed using only manipulations of the bit representations of the entries of the initial state matrix T, and they can be computed in any order. Furthermore, the intermediate state matrices do not even need to be calculated, so no big integer arithmetic is required in the matrix phase. The output of the matrix phase only needs to include the transition matrices and the minimal state matrix.

Note that traditionally, the Montgomery ladder is presented in a slightly different, but equivalent manner. The differnce is that intermediate state matrices even and odd entries are occasionally swapped as needed, so that the transition operations depend on only a single bit in the representation, not two consecutive bits. The reason for us to use a slightly different form is to extend the algorithm to $d > 1$ consistently with the methods already described.

The first step of extending the simplification to $d > 1$, is to recognize that the value entries in each column of the s^{th} intermediate state matrix may be represented by the bit values $(a_s, a_{s+1}, \dots)$ in the notation of Theorem 13. The reduction of the intermediate state matrix modulo two, a binary matrix, together with the information $(a_s, a_{s+1}, \dots)$, completely determines the full value of the intermediate state matrix. Therefore we may equivalent represent the state matrices as a pair (A, B) of binary matrices, where $A = (a_{j,k})$ encodes values like a_s and $B = (b_{j,k})$ encodes the modulo two values of the state matrix. More precisely for state matrix $S = (s_{j,k})$, we have

$$s_{j,k} = 2a_{0,k}(1 - b_{j,k}) + b_{j,k} + 2a_{1,k} + 4a_{2,k} + 2^3 a_{3,k} + \dots \tag{33}$$

On the basis of Theorems 12 and 13, when transitioning to a smaller matrix, the effect on the A component of the state is precisely deletion of the zeroth row. Although the B component is itself a state matrix, the effect on B depends on A.

To determine the effect on B, when transitioning down, we review how Theorem 4 takes a state matrix T and determines the unique transition matrix M and smaller state matrix S such that $T = MS$. The first step of the proof is to halve T_0 and examine the parity. We have $\frac{1}{2}t_{0,k} = a_{0,k} + a_{1,k} + 2a_{2,k} + \dots$. Therefore h is the number of k such that $a_{0,k} + a_{1,k}$ is odd, and $S_h = \frac{1}{2}T_0$. Let B' be the B component of S. Then we have $b'_{h,k} \equiv a_{0,k} + a_{1,k} \bmod 2$. We may write this $B'_h \equiv A_0 + A_1 \bmod 2$.

We have determined M_0 and S_h and B'_h, and next is to determine M_1 and $S_{h\pm1}$ and $B'_{h\pm1}$, for some choice of $h \pm 1$. To do this, we consider $S_{h\pm1} = T_1 - S_h = T_1 - T_0 + S_h$. But $(T_1 - T_0)_k = (2a_{0,k} - 1)(b_{1,k} - b_{0,k}) = 2a_{0,k} - 1b_{1,k}$. Now $2a_{0,k} - 1 \in \{-1, 1\}$, and there is a unique value of k such that $b_{1,k} = 1$. Now the next row to consider is $h' = h + 1$ if, for this unique k, we have $b'_{h,k} = 0$ and otherwise the next row to consider to consider is $h' = h - 1$. In either case, we have $B'_{h'} \equiv B'_h + B_1 \bmod 2$.

Continuing, we just imitate the proof of Theorem 4, except that now the decisions made in that proof using rows of T and S are replaced by decision made using rows of B and B'. This can be done, since only the parities of the differences of rows of T and S are used to determine M and S.

When implementing the algorithm, it is not necessary to actually the transition matrices M as matrices per se, because most the entries are 0. Instead, one can compute the values of f and g for each row, so that $M_j = e_f + e_g$. In the description of the algorithm the f values are put into a matrix F and the g values are put into a matrix G.

3.4 The Point Addition Phase

A row $(r_1, \dots, r_d)$ of a state will represent a point $r_1G_1 + \dots + r_dG_d$ whose x-coordinate we may compute. The difference of any two rows of a state has the form $(l_1, \dots, l_d)$ where $l_1, \dots, l_d \in \{-1, 0, 1\}$. The Montgomery point addition formula will be used, with the aid of computation of $l_1G_1 + \dots + l_dG_d$ when needed. To compute a combination $k_1G + \dots + k_dG$, we first find a state with $(k_1, \dots, k_d)$ as a row. From this state, we will find a sequence successfully smaller states, related transition matrices, which are defined next.

Therefore to compute $r_1G_1 + \dots + r_dG_d$. Find a state matrix T with $R = (r_1, \dots, r_d)$ as a row. It will be convenient to write $G = (G_1, \dots, G_d)$, and to write $RG = r_1G_1 + \dots + r_dG_d$, for any such row vector R in general.

Now factor T per Theorem 8, which can be done efficiently. For each of the intermediate state matrix S in the factorization, and each row S_j of the state matrix, we will calculate the elliptic curve point S_jG. Now S is obtained from a smaller state, say U, and $S_j = U_f + U_g$ for some rows of U_f and U_g of U. We calculate $S_jG = U_fG + U_gG$.

Acknowledgements

Adrian Antipa, René Struik, Rob Lambert, and Scott Vanstone provided extremely helpful discussions of this algorithm.

References

To be added.