Hybrid Binary-Ternary Joint Sparse Form and its Application in Elliptic Curve Cryptography

Jithra Adikari, *Student Member, IEEE,* Vassil Dimitrov, and Laurent Imbert

Abstract

Multi-exponentiation is a common and time consuming operation in public-key cryptography. Its elliptic curve counterpart, called multi-scalar multiplication is extensively used for digital signature verification. Several algorithms have been proposed to speed-up those critical computations. They are based on simultaneously recoding a set of integers in order to minimize the number general multiplications or point additions. When signed-digit recoding techniques can be used, as in the world of elliptic curves, Joint Sparse Form (JSF) and interleaving w-NAF are the most efficient algorithms. The novel recoding algorithm is proposed for a pair of integers based on a decomposition that mixes powers of 2 and powers of 3. It is shown that the so-called Hybrid Binary-Ternary Joint Sparse Form (HBTJSF) is shorter and sparser than the JSF and the interleaving w-NAF. The advantages of the HBTJSF are illustrated for elliptic curve double-scalar multiplication; the operation counts shows a gain of up to 18%.

Index Terms

Multi-exponentiation, Multi-scalar multiplication, Joint sparse form, Binary-ternary number system, Elliptic curves.

I. Introduction

Multi-exponentiation is a common operation in public-key cryptography. Most digital signatures are verified by evaluating an expression of the form $g^a h^b$, where g, h are elements of a multiplicative group; typically the group $\mathbb{F}_q^*$ of non-zero elements of the finite field $\mathbb{F}_q$. To speed-up this operation, one can compute the well known Shamir's trick (see [1] and [2]), which is based on the simple fact it is unnecessary to compute the two expressions separately since only the product is needed. Shamir first suggested to apply the square-and-multiply algorithm to the binary expansions of both a and b at the same time, and further noticed that some extra savings can be obtained by precomputing the product gh. If t denote the bit-length of the largest exponent, this method requires t squarings and $3t/4$ multiplications on average.

This work was supported by the Natural Sciences & Engineering Research Council of Canada.

J. Adikari and V. Dimitrov are with the Advanced Technology Information Processing System Labs. (ATIPS), Department of Electrical and Computer Engineering at the University of Calgary, 2500 University Dr. NW, Calgary, AB T2N 1N4, Canada (e-mail: jithra.adikari@atips.ca; dimitrov@atips.ca).

L. Imbert is with the CNRS "PIMS-Europe" (UMI 3069), Centre for Information Security and Cryptography (CISaC), Department of Mathematics & Statistics at the University of Calgary, 2500 University Dr. NW, Calgary, AB, T2N 1N4, Canada (e-mail: Laurent.Imbert@ucalgary.ca)

In the world of elliptic curves, the same critical operation rewrites $[k]P + [l]Q$, where k and l are two positive integers, and P, Q are two elements of the group of points of an elliptic curve; a nice group where elements can be easily inverted (the cost of computing $-P$ from P is negligible). Naturally, joint signed binary expansions [3], with digits in $\{-1, 0, 1\}$ have been considered. The scalars k, l can be represented as a $2 \times t$ matrix

$$\begin{array}{rl} k = & (k_{t-1} \quad \ldots \quad k_1 \quad k_0) \\ l = & (l_{t-1} \quad \ldots \quad l_1 \quad l_0), \end{array}$$

with $k_i, l_i \in \{-1, 0, 1\}$ for all i. The number of additions required by Shamir's simultaneous method is equal to the so-called joint Hamming weight; i.e., the number of non-zero columns. For example, if k and l are both written in the Non-Adjacent Form [4], [5], the computing $[k]P + [l]Q$ costs $t + 1$ doublings and $5t/9$ additions on average.

Example 1: The $2 \times t$ matrix given by the NAFs of $k = 145$ and $l = 207$

$$\begin{array}{rl} 145 = & (0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1) \\ 207 = & (1 \quad 0 \quad \bar{1} \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad \bar{1}) \end{array}$$

has joint Hamming weight 5.

In [6], Solinas introduced the Joint Sparse Form (JSF) to further reduce the average number of non-zero columns. The main idea behind Solinas' algorithm is to make sure that out of three consecutive columns, at least one is a zero-column. Solinas' algorithm is given in terms of arithmetic operations but it basically reduces to computations modulo 8 (bit operations). By carefully choosing the positive/negative values of the remainders (mod 8), Solinas proves the uniqueness and optimality (in the context of joint signed binary expansions) of the JSF, showing that the computation of $[k]P + [l]Q$ requires $t + 1$ doublings and $t/2$ additions on average.

Example 2: Using the same values as above ($k = 145$, $l = 207$), the JSF

$$\begin{array}{rl} 145 = & (1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1) \\ 207 = & (1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad \bar{1}) \end{array}$$

has Hamming weight 4.

The simultaneous methods described above require precomputations of points involving both P and Q. For example, the JSF algorithm needs the points $P + Q$ and $P - Q$ to be precomputed. On the other hand, interleaving methods use precomputed values that only involve a single point, which allows to use different methods for each precomputed point (such as different width-w NAFs); the doubling steps being done jointly. The overall cost of interleaving methods is $t + 1$ doublings and $2t/(w + 1)$ additions on average (see [2, pp 111–113] for more details). We give an example of interleaving w-NAF in Section III.

In this paper, we describe a novel joint recoding scheme which uses both bases 2 and 3 in order to reduce the average number of non-zero columns. In Section II, we present the basics of the so-called hybrid binary-ternary number systems. In Section III, we extend the concept to represent pairs of integers and we introduce a new joint recoding algorithm. We analyzes our algorithm in Section IV and present some numerical results in Section V.

II. Hybrid Binary-Ternary Number System

The Hybrid Binary-Ternary Number System (HBTNS) was introduced by Dimitrov and Cooklev in [7] for speeding-up modular exponentiation. In this system, an integer is written as a sum of powers of 2 and powers of 3; i.e. it mixes bits and trits (radix-3 digits) except that the digit 2 never occurs. The use of base 3 naturally reduces the number of digits[1] required to represent a t-bit integer. In fact, it can be shown that a t-bit number can be written with $\approx 0.88058t$ digits, whereas the average base is ≈ 2.19617 (see [7] for more details). More importantly, this number system is also very sparse; the average number of non-zero digits in HBTNS is $\approx 0.3381t$. Algorithm 1 can be used to calculate the HBTNS representation of a positive integer.

Algorithm 1 HBTNS representation

Input : An integer $n > 0$

Output : Arrays `digits[], base[]`

```
i = 0
while n > 0 do
    if n ≡ 0 (mod 3) then
        base[i] = 3; digits[i] = 0; n = n/3;
    else if n ≡ 0 (mod 2) then
        base[i] = 2; digits[i] = 0; n = n/2;
    else
        base[i] = 2; digits[i] = 1; n = n/2;
    end if
    i = i + 1
end while
return digits[], base[]
```

Example 3: The hybrid binary-ternary representation of $n = 703 = (1010111111)_2$

$$\texttt{digits[]} = [1, 0, 0, 0, 1, 0, 0, 1]$$

$$\texttt{base[]} = [2, 3, 3, 3, 2, 3, 2, 2].$$

has only 8 digits among which 3 only are non-zero. Note that the binary representation requires 10 bits, out of which 8 are different from zero.

The idea of mixing bases 2 and 3 for elliptic curve scalar multiplication has been proposed by Ciet et al. in [8] using the same decomposition as in Algorithm 1. Dimitrov, Imbert and Mishra generalized this concept in [9] by

[1] Although we only deal with 0s and 1s, the term ”digit” is more appropriate than ”bit” because of the use of base 3.

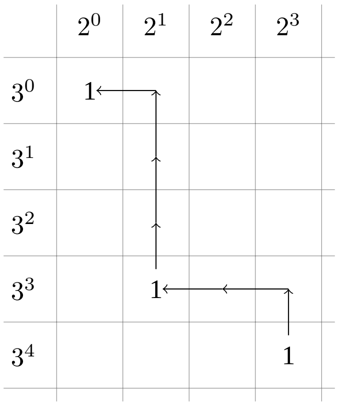

Fig. 1. An example of staircase walk for a double-base chain representing 703

using a greedy approach to compute special signed double-base expansions; i.e. expressions of the form

$$\sum_i \pm 2^{a_i} 3^{b_i}, \text{ with } a_i, b_i \geq 0,$$

where the exponents form two simultaneously decreasing sequences. These expansions, called double-base chains (see Def. 1 below), allows for fast scalar multiplication. See [10] for more details about this number system.

Definition 1 (Double-base chain): Given $k > 0$, a sequence $(K_n)_{n>0}$, of positive integers satisfying: $K_1 = 1$, $K_{n+1} = 2^u 3^v K_n + s$, with $s \in \{-1, 1\}$ for some $u, v \geq 0$, and such that $K_m = k$ for some $m > 0$, is called a double-base chain for k. The length, m, of a double-base chain is equal to the number of terms (often called $\{2, 3\}$-integers), used to represent k.

Any elliptic curve scalar multiplication algorithm based on mixing powers of 2 and powers of 3 requires point doublings and additions, as well as, possibly fast, point triplings. In [9], Dimitrov et al. also proposed an efficient tripling formula in Jacobian coordinates for ordinary elliptic curves over large prime fields (see [11] for improved formulas). In [12], Doche and Imbert further extended the idea by allowing larger digits sets as in the w-NAF algorithms.

An easy way to visualize expansions using two bases (say e.g. 2 and 3), is to use a two-dimensional array (the columns represent the powers of 2 and the rows represent the powers of 3) into which each non-zero cell contains the sign of the corresponding term. (by convention, the upper-left corner corresponds to $2^0 3^0 = 1$.) A double-base chain can thus be represented by a staircase walk from the bottom-right corner to the upper-left corner, with non-zero digits distributed along this path. An example of such a double-base chain is shown in Fig. 1; it was obtained using Algorithm 1. (Since a given set of non-zero cells can lead to many different staircase walks, we adopt the convention to walk North as much as we can before going East.)

In the next section, we present an algorithm which computes two double-base chains that share the same staircase walk; only the distribution of the digits along the path differ.

III. HYBRID BINARY TERNARY JOINT SPARSE FORM

In Algorithm 2 below, the hybrid binary-ternary joint sparse form of a pair of integers is calculated by first checking whether both k_1 and k_2 are divisible by 3. If this is the case, both digits are set to 0 and the base set to 3, otherwise we check whether they are both divisible by 2 and proceed accordingly. Finally, if the pair is not divisible by 2 and 3, we make both numbers divisible by 6 by subtracting k_i mods $6 \in \{-2, -1, 0, 1, 2, 3\}$ from k_i, and then divide the results by 2. Therefore, in the next step, both numbers are divisible by 3 and we generate a zero column.

Algorithm 2 Hybrid binary-ternary joint sparse form (HBTJSF)

Input : Two positive integers k_1, k_2

Output : Arrays `hbt1[], hbt2[], base[]`

$i = 0$
while $k_1 > 0$ or $k_2 > 0$ **do**
  if $k_1 \equiv 0 \pmod 3$ and $k_2 \equiv 0 \pmod 3$ **then**
    `base[`i`]` = 3;
    `hbt1[`i`]` = `hbt2[`i`]` = 0;
    $k_1 = k_1/3$; $k_2 = k_2/3$;
  else if $k_1 \equiv 0 \pmod 2$ and $k_2 \equiv 0 \pmod 2$ **then**
    `base[`i`]` = 2;
    `hbt1[`i`]` = `hbt2[`i`]` = 0;
    $k_1 = k_1/2$; $k_2 = k_2/2$;
  else
    `base[`i`]` = 2;
    `hbt1[`i`]` = k_1 mods 6; `hbt2[`i`]` = k_2 mods 6;
    $k_1 = (k_1 -$ `hbt1[`i`]`$)/2$; $k_2 = (k_2 -$ `hbt2[`i`]`$)/2$;
  end if
  $i = i + 1$
end while
return `hbt1[], hbt2[], base[]`

Example 4: The following example illustrates the advantage of the HBTJSF. For $k_1 = 1225$ and $k_2 = 723$, the Joint Sparse Form

$$\begin{aligned} 1225 &= (1 \quad 0 \quad 1 \quad 0 \quad \bar{1} \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1) \\ 723 &= (1 \quad 0 \quad \bar{1} \quad 0 \quad 0 \quad \bar{1} \quad 0 \quad \bar{1} \quad \bar{1} \quad 0 \quad \bar{1}) \end{aligned}$$

has joint Hamming weight 7. In the interleaving method, the non-zero elements in both w-NAF representations are

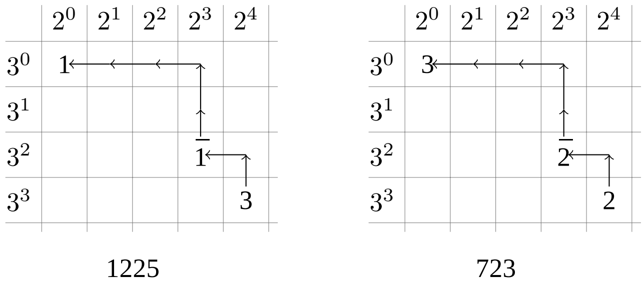

Fig. 2. Double-base chains for 1225 and 723

	P	$2P$	$3P$
Q	$P \pm Q$	$2P \pm Q$	$3P \pm Q$
$2Q$	$P \pm 2Q$	-	$3P \pm 2Q$
$3Q$	$P \pm 3Q$	$2P \pm 3Q$	-

TABLE I

PRECOMPUTATION MATRIX FOR HBTJSF POINT MULTIPLICATION

considered, instead of joint hamming weight. The interleaving w-NAF (with $w = 5$ for 1225 and $w = 4$ for 723)

$$\begin{aligned} \text{5-NAF}(1225) &= (1 \quad 0 \quad 0 \quad 0 \quad 0 \quad -13 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 9) \\ \text{4-NAF}(723) &= (0 \quad 0 \quad 0 \quad 3 \quad 0 \quad 0 \quad 0 \quad -3 \quad 0 \quad 0 \quad 0 \quad 3) \end{aligned}$$

has 6 non-zero elements (using $w = 4$ for 1225 and $w = 5$ for 723 also leads to 6 non-zero elements). Using Algorithm 2, the hybrid binary-ternary joint sparse form

$$\begin{aligned} 1225 &= (3 \quad 0 \quad \bar{1} \quad 0 \quad 0 \quad 0 \quad 0 \quad 1) \\ 723 &= (2 \quad 0 \quad \bar{2} \quad 0 \quad 0 \quad 0 \quad 0 \quad 3) \\ \texttt{base[]} &= (2 \quad 3 \quad 2 \quad 2 \quad 2 \quad 3 \quad 3 \quad 2) \end{aligned}$$

only requires 8 digits and has joint Hamming weight 3. We show the corresponding double-base chains in Fig. 2. Note the digits are distributed along the same staircase walk.

Since the HBTJSF uses the digit set $\{-2, -1, 0, 1, 2, 3\}$, some points have to be precomputed and stored (see Section V for a discussion regarding these precomputations). Table I gives the precomputation matrix. Since the negation of a point is negligible, only one set of point difference need to be calculated. Others can be calculated online. For example, $P - 2Q$ can be used to calculate, $2Q - P$.

IV. THEORETICAL ANALYSIS

Let us analyzes the behaviour of Algorithm 2. We consider numbers of the form $6k + j$ with $j \in \{-2, \ldots, 3\}$. We want to know how often our algorithm performs a division by 3 and how often it performs a division by 2. More importantly, we want to know how many non-zero columns we can expect on average.

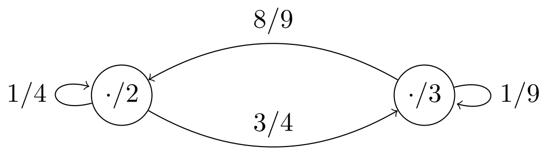

Fig. 3. State diagram illustrating divisions by 2 and divisions by 3 in Algorithm 2

Only the numbers of the form $6k$ and $6k+3$ are divisible by 3. Therefore, since both k_1 and k_2 have to be divisible by 3, our algorithm performs a division by 3 with probability $1/9$. Obviously, we divide by 2 in all the other cases; i.e., with probability $8/9$. These are the initial probabilities. Now, divisions by 2 either occur when both numbers are divisible by 2 (numbers of the for $6k-2$, $6k$ or $6k+2$); i.e., with probability $1/4$, or when we make both numbers divisible by 6, which therefore occurs with probability $3/4$. This Markov process can be illustrated by a state diagram (see Fig. 3).

Using classic matrix diagonalization, the corresponding transition matrix

$$P = \begin{pmatrix} 1/4 & 3/4 \\ 8/9 & 1/9 \end{pmatrix}$$

can be decomposed into the form

$$P = R \times \begin{pmatrix} 1 & 0 \\ 0 & -23/36 \end{pmatrix} \times R^{-1},$$

where

$$R = \frac{1}{59}\begin{pmatrix} 32 & 27 \\ 32 & -32 \end{pmatrix} \quad \text{and} \quad R^{-1} = \frac{1}{32}\begin{pmatrix} 32 & 27 \\ 32 & -32 \end{pmatrix}.$$

The average probabilities π_∞ are obtained as $\lim_{n\to\infty} \pi_0 P^n$, with $\pi_0 = (8/9, 1/9)$ our initial probabilities. We have

$$\pi_n = \pi_0 P^n = (32/59 + 184/531(-23/36)^n, 27/59 + 184/531(-23/36)^n)$$

and therefore

$$\pi_\infty = \left(\frac{32}{59}, \frac{27}{59}\right).$$

Hence, we divide by 2 with probability $32/59 \approx 0.542$ and we divide by 3 with probability $27/59 \approx 0.458$. Using these probabilities, we can evaluate the average base

$$\beta = \sqrt[59]{2^{32}3^{27}} = \sqrt[59]{32751691810479015985152} \approx 2.407765$$

For a pair of t-bit integers, the average digit length of the HBTJSF array is thus approximately

$$(\log_\beta 2) \times t \approx 0.7888 \times t. \tag{1}$$

To complete our analysis, we need to know how many non-zero columns we can expect on average. Using the same reasoning as above, we have a zero column when both numbers are divisible by 3 or when both numbers

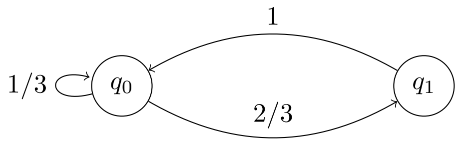

Fig. 4. State diagram illustrating the probabilities of zero columns (state q_0) and non-zero columns (state q_1) in Algorithm 2

TABLE II

THEORETICAL COMPARISON OF HBTJSF, JSF AND INTERLEAVING w-NAF FOR A t-BIT PAIR OF INTEGERS

Parameters	HBTJSF	JSF	Interleaving w-NAF
Average base	2.41	2	2
Avg # col.	$0.79t$	$t+1$	$t+1$
Avg # base 2 col.	$0.43t$	$t+1$	$t+1$
Avg # base 3 col.	$0.36t$	0	0
Avg # non-zero col.	$0.32t$	$0.5t$	$2t/(w+1)$
Precomp.	18	2	$2^{w-1}-2$

are divisible by 2, which occurs with probability $1/9 + 1/4 - 1/36 = 1/3$ (numbers simultaneously divisible by 6 need only be counted once). Reversely, we have a non-zero column with probability $2/3$. Since our algorithm never generates two consecutive non-zero columns, we obtain the state diagram illustrated in Fig. 4. Using the same analysis as above, the transition matrix

$$Q = \begin{pmatrix} 1/3 & 2/3 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & -2/3 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -2/3 \end{pmatrix} \begin{pmatrix} 3/5 & 2/5 \\ -3/5 & 3/5 \end{pmatrix}$$

allows us to compute the average number of columns of each type. We have $\pi_\infty = (3/5, 2/5)$; meaning that, on average, our algorithm generates a non-zero column with probability $2/5$. The expected number of elliptic curve additions can thus be derived from (1). We have

$$\#\text{Add} \approx \frac{2}{5} \times 0.7888 \times t \approx 0.3155 \times t.$$

We summarized our theoretical results in Table II, with real values rounded to the nearest hundredth. (For simplicity, we consider that the same window width is used for both numbers in the interleaving w-NAF method.)

V. COMPARISONS

Based on the above analysis, we compare our algorithm with the JSF and the w-NAF interleaving method (assuming windows of size $w = 4$). We consider two kinds of curves for which we know that triplings are useful [13]:

- Ordinary elliptic curves over large prime fields with Jacobian coordinates (with $a = -3$),
- tripling-oriented Doche-Icart-Kohel curves [14].

TABLE III

COSTS OF SOME CURVE OPERATIONS FOR ORDINARY ELLIPTIC CURVES OVER PRIME FIELDS IN JACOBIAN COORDINATES ($a = -3$) AND TRIPLING-ORIENTED DIK CURVES

Weierstrass / Jacobian ($a = -3$)

	Cost	S = 0.8M
Doubling	3M + 5S	7M
Tripling	7M + 7S	12.6M
Addition (mixed)	7M + 4S	10.2M

Tripling-oriented DIK

	Cost	S = 0.8M
Doubling	2M + 7S	7.6 M
Tripling	6M + 6S	10.8 M
Addition (mixed)	7M + 4S	10.2 M

TABLE IV

COMPARISONS BETWEEN HBTJSF, JSF AND INTERLEAVING w-NAF FOR 256-BIT INTEGERS

Weierstrass / Jacobian ($a = -3$)

	HBTJSF	JSF	Inter. 4-NAF
Mult. counts for dbl.	770	1799	1799
Mult. counts for tpl.	1159	0	0
Mult. counts for add	836	1311	1049
Total mult. counts	2765	3110	2848
Gain (%)	–	11.09	2.91

Tripling-oriented DIK

	HBTJSF	JSF	Inter. 4-NAF
Mult. counts for dbl.	836	1953	1953
Mult. counts for tpl.	994	0	0
Mult. counts for add	836	1311	1049
Total mult. counts	2666	3264	3002
Gain (%)	–	18.32	11.19

The costs of the necessary curve operations are given in Table III; the last column give equivalent multiplication counts assuming $S = 0.8M$. These costs are reported in the comprehensive and accurate explicit-formulas database [11]. Our results are summarized in Table IV for 256-bit pairs of integers (similar savings can be observed for other sizes).

For our operation counts, we have omitted the costs of precomputations. In the case of JSF, only $P+Q$ and $P-Q$ have to be computed, which is (almost, but not exactly) equivalent to two additions. In the case of interleaving w-NAF, we have to precompute $3P$, $3Q$, $5P$, $5Q$, $\dots$, $(2^{w-1}-1)P$, $(2^{w-1}-1)Q$; i.e., a total of $2^{w-1}-2$ points. The exact operation counts depends on the way those computations are implemented. In the case of HBTJSF, 18 points are needed as shown in Table I. However, it is quite unlikely that all the combinations will be required in the process of computing a double-scalar multiplication. Therefore, a better option might be to perform those computations online when required. Of course, as soon as one of those points is encountered, it must be stored for future use. But even if all 18 points are precomputed beforehand, the exact operation counts is less than the cost of 2 doublings plus 16 additions, since common subexpressions eliminations techniques can be considered.

VI. Conclusions

A new recoding algorithm for a pair of integers has been proposed. It is based on a decomposition of two integers using mixed powers of 2 and 3. Our analysis shows that it requires almost 20% fewer digits than the binary representation and that the average ratio of non-zero columns over digit length is $2/5$. We have illustrated the advantages of the so-called HBTJSF for elliptic curve double-scalar multiplication. Compared to the commonly used JSF and interleaving w-NAF methods, the savings obtained with HBTJSF are significant (up to 18%) for curves for which triplings are useful, such as e.g. ordinary curves over large prime fields or tripling-oriented DIK curves.

References

[1] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, July 1985.

[2] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[3] A. D. Booth, “A signed binary multiplication technique,” *Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951, reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 1, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[4] G. W. Reitwiesner, “Binary arithmetic,” *Advances in Computers*, vol. 1, pp. 231–308, 1960.

[5] M. Joye and S.-M. Yen, “Optimal left-to-right binary signed-digit exponent recoding,” *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 740–748, 2000.

[6] J. A. Solinas, “Low-weight binary representations for pairs of integers,” Center for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada, Research report CORR 2001-41, 2001.

[7] V. Dimitrov and T. V. Cooklev, “Two algorithms for modular exponentiation based on nonstandard arithmetics,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science*, vol. E78-A, no. 1, pp. 82–87, Jan. 1995, special issue on cryptography and information security.

[8] M. Ciet, M. Joye, K. Lauter, and P. L. Montgomery, “Trading inversions for multiplications in elliptic curve cryptography,” *Designs, Codes and Cryptography*, vol. 39, no. 2, pp. 189–206, May 2006.

[9] V. Dimitrov, L. Imbert, and P. K. Mishra, “Efficient and secure elliptic curve point multiplication using double-base chains,” in *Advances in Cryptology, ASIACRYPT’05*, ser. Lecture Notes in Computer Science, vol. 3788. Springer, 2005, pp. 59–78.

[10] ——, “The double-base number system and its application to elliptic curve cryptography,” *Mathematics of Computation*, vol. 77, no. 262, pp. 1075–1104, 2008.

[11] D. J. Bernstein and T. Lange, “Explicit-formulas database,” URL: http://www.hyperelliptic.org/EFD/, joint work by Daniel J. Bernstein and Tanja Lange, building on work by many authors.

[12] C. Doche and L. Imbert, “Extended double-base number system with applications to elliptic curve cryptography,” in *Progress in Cryptology, INDOCRYPT’06*, ser. Lecture Notes in Computer Science, vol. 4329. Springer, 2006, pp. 335–348.

[13] D. J. B. andP. Birkner, T. Lange, and C. Peters, “Optimizing double-base elliptic-curve single-scalar multiplication,” in *Progress in Cryptology – INDOCRYPT2007*, ser. Lecture Notes in Computer Science, vol. 4859. Springer, 2007, pp. 167–182.

[14] C. Doche, T. Icart, and D. R. Kohel, “Efficient scalar multiplication by isogeny decompositions,” in *Public Key Cryptography, PKC’06*, ser. Lecture Notes in Computer Science, vol. 3958. Springer, 2006, pp. 191–206.