The Double-Base Number System in Elliptic Curve Cryptograhy

Christophe Doche[1] Laurent Imbert[2]

[1]Macquarie University, Sydney, Australia

[2]CNRS-PIMS, University of Calgary, Canada

Asilomar Conference on Signal, Systems and Computers, 2008

From joint works with: Jithra Adikari, Vassil Dimitrov, Fabrice Philippe, David Kohel, Francesco Sica

Motivations

Fast exponentiation: Given $(G, \times)$, $g \in G$ and $n \geq 0$, compute g^n.

Elliptic curve scalar multiplication: Given P on an elliptic curve, and $k \geq 0$, compute $[k]P = P + P + \cdots + P$ (k times).

This operation is the most time consuming in elliptic curve protocols (ECDH, ECDSA, etc).

How quickly can we do this?

Important variant of the problem, multi-scalar multiplication: $k_1, k_2, P, Q \to k_1 P + k_2 Q$, important operation in elliptic curve signature verification.

Point multiplication algorithms & addition chains

Double-and-add: $k = \sum_{i=0}^{n-1} k_i 2^i$, with $k_i \in \{0, 1\}$

$n - 1$ DBL, $n/2$ ADD on average

$1717 = 11010110101$

1 2 3 6 12 13 26 52 53 106 107 214 428 429 858 1716 1717

Point multiplication algorithms & addition chains

Double-and-add: $k = \sum_{i=0}^{n-1} k_i 2^i$, with $k_i \in \{0, 1\}$

$n - 1$ DBL, $n/2$ ADD on average

$1717 = 11010110101$

1 2 3 6 12 13 26 52 53 106 107 214 428 429 858 1716 1717

Signed digits: Canonic SD, NAF, $k_i \in \{\bar{1}, 0, 1\}$

n DBL, $n/3$ ADD on average

$\mathrm{NAF}(1717) = 100\bar{1}0\bar{1}0\bar{1}0101$

1 2 4 8 7 14 28 27 54 108 107 214 428 429 858 1716 1717

Point multiplication algorithms & addition chains

Double-and-add: $k = \sum_{i=0}^{n-1} k_i 2^i$, with $k_i \in \{0, 1\}$

$n - 1$ DBL, $n/2$ ADD on average

$1717 = 11010110101$

1 2 3 6 12 13 26 52 53 106 107 214 428 429 858 1716 1717

Signed digits: Canonic SD, NAF, $k_i \in \{\bar{1}, 0, 1\}$

n DBL, $n/3$ ADD on average

$\mathrm{NAF}(1717) = 100\bar{1}0\bar{1}0\bar{1}0101$

1 2 4 8 7 14 28 27 54 108 107 214 428 429 858 1716 1717

Window methods: wNAF, $|k_i| < 2^{w-1}$ (processes w digits at a time)

n DBL, $n/(w+1)$ ADD on average + precomp.

$3\mathrm{NAF}(1717) = 300300\bar{1}00\bar{3}$

3 6 12 24 27 54 108 216 215 430 860 1720 1717

$4\mathrm{NAF}(1717) = 7000\bar{5}0005$

7 14 28 56 112 107 214 428 856 1712 1717

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing* k.

1 2 4 8 16 48 144 143 286 572 1716 1717

1						

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing k*.

1 2 4 8 16 48 144 143 286 572 1716 1717

	1					

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing* k.

1 2 4 8 16 48 144 143 286 572 1716 1717

		1				

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing* k.

1 2 4 8 16 48 144 143 286 572 1716 1717

			1			

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing k*.

1 2 4 8 16 48 144 143 286 572 1716 1717

				1		

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing* k.

1 2 4 8 16 48 144 143 286 572 1716 1717

				1		

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing k*.

1 2 4 8 16 48 144 143 286 572 1716 1717

				1		

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing* k.

1 2 4 8 16 48 144 143 286 572 1716 1717

-1						
				1		

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing* k.

1 2 4 8 16 48 144 143 286 572 1716 1717

	-1					
					1	

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing k*.

1 2 4 8 16 48 144 143 286 572 1716 1717

		-1				
						1

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing* k.

1 2 4 8 16 48 144 143 286 572 1716 1717

		-1				
						1

Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing k*.

1 2 4 8 16 48 144 143 286 572 1716 1717

1						
		-1				
						1

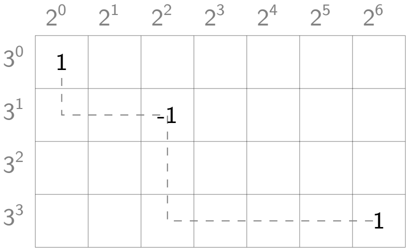
Double-base numbers and chains

Given $k > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + d_i, \text{ with } d_i \in \{-1, 1\}$$

for some $u_i, v_i \geq 0$, and such that $C_n = k$ for some $n > 0$, is called a *double-base chain computing* k.

1 2 4 8 16 48 144 143 286 572 1716 1717



More formally:

$$k = \sum_{i=1}^{n} d_i 2^{a_i} 3^{b_i}, \quad d_i \in \{-1, 1\}$$

with $(a_i, b_i) \searrow$

Computing double-base chains

A greedy approach:

$s \leftarrow 1$
while $k \neq 0$ **do**
  find the best approximation of k of the form $z = 2^a 3^b$ with $0 \leq a \leq \mathcal{A}$ and $0 \leq b \leq \mathcal{B}$
  output term (s, a, b); $\mathcal{A} \leftarrow a$; $\mathcal{B} \leftarrow b$
  if $k < z$ then $s \leftarrow -s$
  $k \leftarrow |k - z|$

Computing double-base chains

A greedy approach:

$s \leftarrow 1$
while $k \neq 0$ **do**
  find the best approximation of k of the form $z = 2^a 3^b$ with $0 \leq a \leq \mathcal{A}$ and $0 \leq b \leq \mathcal{B}$
  output term (s, a, b); $\mathcal{A} \leftarrow a$; $\mathcal{B} \leftarrow b$
  if $k < z$ then $s \leftarrow -s$
  $k \leftarrow |k - z|$

$21687 \quad =$

	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
3^0									
3^1									
3^2									
3^3									
3^4									

Computing double-base chains

A greedy approach:

$s \leftarrow 1$
while $k \neq 0$ **do**
    find the best approximation of k of the form $z = 2^a 3^b$ with $0 \leq a \leq \mathcal{A}$ and $0 \leq b \leq \mathcal{B}$
    output term (s, a, b); $\mathcal{A} \leftarrow a$; $\mathcal{B} \leftarrow b$
    if $k < z$ then $s \leftarrow -s$
    $k \leftarrow |k - z|$

$$21687 \quad = \quad 20736 \;\big|\; (951)$$

	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
3^0									
3^1									
3^2									
3^3									
3^4									1

Computing double-base chains

A greedy approach:

$s \leftarrow 1$
while $k \neq 0$ **do**
  find the best approximation of k of the form $z = 2^a 3^b$ with $0 \leq a \leq \mathcal{A}$ and $0 \leq b \leq \mathcal{B}$
  output term (s, a, b); $\mathcal{A} \leftarrow a$; $\mathcal{B} \leftarrow b$
  if $k < z$ then $s \leftarrow -s$
  $k \leftarrow |k - z|$

$$\begin{array}{rcr|r} 21687 & = & 20736 & (951) \\ & + & 864 & (87) \end{array}$$

	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
3^0									
3^1									
3^2									
3^3						1			
3^4									1

Computing double-base chains

A greedy approach:

$s \leftarrow 1$
while $k \neq 0$ **do**
  find the best approximation of k of the form $z = 2^a 3^b$ with $0 \leq a \leq \mathcal{A}$ and $0 \leq b \leq \mathcal{B}$
  output term (s, a, b); $\mathcal{A} \leftarrow a$; $\mathcal{B} \leftarrow b$
  if $k < z$ then $s \leftarrow -s$
  $k \leftarrow |k - z|$

$$\begin{array}{rcr|r} 21687 & = & 20736 & (951) \\ & + & 864 & (87) \\ & + & 96 & (-9) \end{array}$$

	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
3^0									
3^1						1			
3^2									
3^3						1			
3^4									1

Computing double-base chains

A greedy approach:

$s \leftarrow 1$
while $k \neq 0$ **do**
    find the best approximation of k of the form $z = 2^a 3^b$ with $0 \leq a \leq \mathcal{A}$ and $0 \leq b \leq \mathcal{B}$
    output term (s, a, b); $\mathcal{A} \leftarrow a$; $\mathcal{B} \leftarrow b$
    if $k < z$ then $s \leftarrow -s$
    $k \leftarrow |k - z|$

$$\begin{array}{rcr|r} 21687 & = & 20736 & (951) \\ & + & 864 & (87) \\ & + & 96 & (-9) \\ & - & 8 & (-1) \end{array}$$

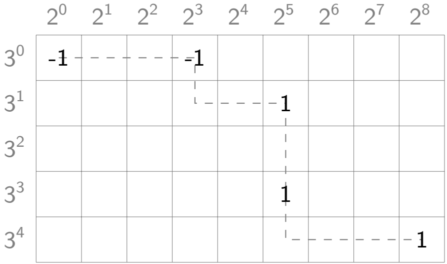
	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
3^0				-1					
3^1						1			
3^2									
3^3						1			
3^4									1

Computing double-base chains

A greedy approach:

$s \leftarrow 1$
while $k \neq 0$ **do**
  find the best approximation of k of the form $z = 2^a 3^b$ with $0 \leq a \leq \mathcal{A}$ and $0 \leq b \leq \mathcal{B}$
  output term (s, a, b); $\mathcal{A} \leftarrow a$; $\mathcal{B} \leftarrow b$
  if $k < z$ then $s \leftarrow -s$
  $k \leftarrow |k - z|$

$$
\begin{array}{rcr|r}
21687 & = & 20736 & (951) \\
 & + & 864 & (87) \\
 & + & 96 & (-9) \\
 & - & 8 & (-1) \\
 & - & 1 & (0)
\end{array}
$$



Length a db-chain = # non-zero terms = # curve additions

Length of a double-base chain

The greedy algorithm does not produce optimal chains. This approach, however, has some interests for elliptic curves with fast tripling such as ordinary curves (over $\mathbb{F}_p$) or DIK3 curves.

We know how to compute the length of the shortest (unsigned) db-chain for k.

Size of k (in bits)	greedy		optimal	
	unsigned	signed	unsigned	signed
64	26.09	18.55	17.22	?
128	54.52	34.88	33.27	?
160	72.21	44.96	40.85	?
256	119.26	75.78	64.35	?

Average values for 10000 random integers

There is still room for improvements!

Double-scalar multiplication

Given $k_1, k_2 > 0$ and points P, Q on an elliptic curve, compute $k_1 P + k_2 Q$

Computing $k_1 P$ and $k_2 Q$ independently is not efficient. We use a method known as *"Shamir's trick"*.

Example: $37P + 22Q$

$$\begin{array}{rcllllll} 37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\ 22 & = & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

Double-scalar multiplication

Given $k_1, k_2 > 0$ and points P, Q on an elliptic curve, compute $k_1 P + k_2 Q$

Computing $k_1 P$ and $k_2 Q$ independently is not efficient. We use a method known as *"Shamir's trick"*.

Example: $37P + 22Q$

$$\begin{array}{rcllllll} 37 & = & \quad {\color{red}1} & 0 & 0 & 1 & 0 & 1 \\ 22 & = & \quad 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

P

Double-scalar multiplication

Given $k_1, k_2 > 0$ and points P, Q on an elliptic curve, compute $k_1 P + k_2 Q$

Computing $k_1 P$ and $k_2 Q$ independently is not efficient. We use a method known as *"Shamir's trick"*.

Example: $37P + 22Q$

$$
\begin{array}{rcrrrrrr}
37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\
22 & = & 0 & 1 & 0 & 1 & 1 & 0
\end{array}
$$

P

2P

2P+Q

Double-scalar multiplication

Given $k_1, k_2 > 0$ and points P, Q on an elliptic curve, compute $k_1 P + k_2 Q$

Computing $k_1 P$ and $k_2 Q$ independently is not efficient. We use a method known as *"Shamir's trick"*.

Example: $37P + 22Q$

$$\begin{array}{ccccccccc} 37 & = & & 1 & 0 & 0 & 1 & 0 & 1 \\ 22 & = & & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

P

2P

2P+Q

4P + 2Q

Double-scalar multiplication

Given $k_1, k_2 > 0$ and points P, Q on an elliptic curve, compute $k_1 P + k_2 Q$

Computing $k_1 P$ and $k_2 Q$ independently is not efficient. We use a method known as *"Shamir's trick"*.

Example: $37P + 22Q$

$$\begin{array}{rcllllll} 37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\ 22 & = & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

P 9P+5Q

2P

2P+Q

4P + 2Q

8P + 4Q

Double-scalar multiplication

Given $k_1, k_2 > 0$ and points P, Q on an elliptic curve, compute $k_1 P + k_2 Q$

Computing $k_1 P$ and $k_2 Q$ independently is not efficient. We use a method known as *"Shamir's trick"*.

Example: $37P + 22Q$

$$\begin{array}{rcrrrrrr} 37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\ 22 & = & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

P	9P+5Q
2P	18P+10Q
2P+Q	18P+11Q
4P + 2Q	
8P + 4Q	

Double-scalar multiplication

Given $k_1, k_2 > 0$ and points P, Q on an elliptic curve, compute $k_1 P + k_2 Q$

Computing $k_1 P$ and $k_2 Q$ independently is not efficient. We use a method known as *"Shamir's trick"*.

Example: $37P + 22Q$

$$
\begin{array}{rcrrrrrr}
37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\
22 & = & 0 & 1 & 0 & 1 & 1 & 0
\end{array}
$$

P	9P+5Q
2P	18P+10Q
2P+Q	18P+11Q
4P + 2Q	36P + 22Q
8P + 4Q	37P + 22Q

Double-scalar multiplication

Given $k_1, k_2 > 0$ and points P, Q on an elliptic curve, compute $k_1 P + k_2 Q$

Computing $k_1 P$ and $k_2 Q$ independently is not efficient. We use a method known as *"Shamir's trick"*.

Example: $37P + 22Q$

$$\begin{array}{rcrrrrrr} 37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\ 22 & = & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

P	9P+5Q
2P	18P+10Q
2P+Q	18P+11Q
4P + 2Q	36P + 22Q
8P + 4Q	37P + 22Q

Cost:

$n - 1$ DBL + # non-zero col. ADD

Precomputations: $P, Q, P + Q$

Interleaving methods

How can we reduce the number of non-zero columns?

Using NAF representations for k_1 and k_2 cost n DBL and $5n/9$ ADD on average but it can result in no improvements!

$$\begin{array}{ccccccc} 1 & 0 & \bar{1} & 0 & 1 & 0 & \bar{1} \\ 0 & 1 & 0 & 1 & 0 & \bar{1} & 0 \end{array}$$

The probability of a non-zero column decreases when using wNAF.

It is possible to use windows of different width for k_1 and k_2, as one usually know either P or Q.

Precomputed points only involve one point: $P, 3P, 5P, \ldots, 2^{w_1-1}P$, $Q, 3Q, 5Q, \ldots, 2^{w_2-1}Q$

Cost: n DBL and $\#$ non-zero **digits** ADD

Joint-sparse form

In 2001, Solinas proposed a recoding technique to converts a pair (k_1, k_2) into a so-called *joint-sparse form*. Out of any three consecutive columns, at least one is a zero-column.

$$\begin{array}{rl} 113 = & (1 \quad 0 \quad 0 \quad \bar{1} \quad 0 \quad 0 \quad 0 \quad 1) \\ 203 = & (1 \quad 1 \quad 0 \quad 1 \quad 0 \quad \bar{1} \quad 0 \quad \bar{1}) \end{array}$$

The JSF is computed using basic arithmetic operations (mod 8).

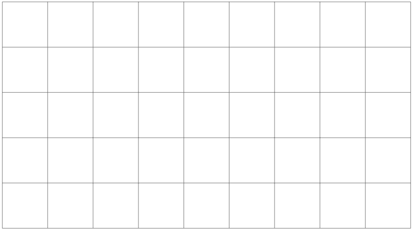
The JSF of a pair of integer is **unique** and **optimal**: every other recoding in $\{-1, 0, 1\}$ requires more non-zero column.

Cost: n DBL + $n/2$ ADD on average.

Precomputations: $P, Q, P+Q, P-Q$

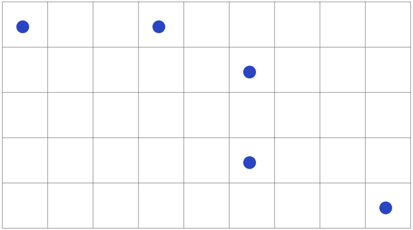
Double-scalar multiplication using double-base chains

The idea is to find two double-base chains which share the same path, with digits possibly appearing at different locations.



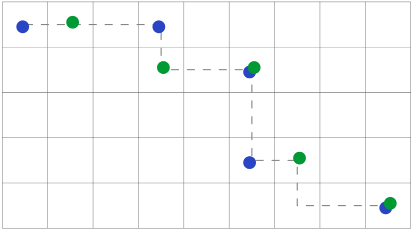
Double-scalar multiplication using double-base chains

The idea is to find two double-base chains which share the same path, with digits possibly appearing at different locations.



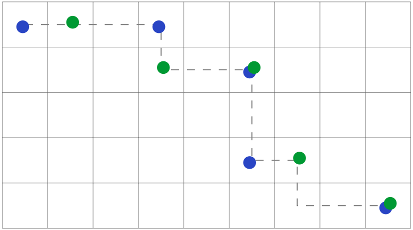
Double-scalar multiplication using double-base chains

The idea is to find two double-base chains which share the same path, with digits possibly appearing at different locations.



Double-scalar multiplication using double-base chains

The idea is to find two double-base chains which share the same path, with digits possibly appearing at different locations.



How do we find such a path and digits?

The hybrid binary-ternary joint sparse form

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

The hybrid binary-ternary joint sparse form

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

Input: $k_1, k_2 > 0$
Output: `hbt1[]`, `hbt2[]`, `base[]`

The hybrid binary-ternary joint sparse form

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

Input: $k_1, k_2 > 0$
Output: `hbt1[]`, `hbt2[]`, `base[]`

1: $i = 0$
2: **while** $k_1 > 0$ or $k_2 > 0$ **do**
3: **if** $k_1 \equiv 0 \pmod 3$ and $k_2 \equiv 0 \pmod 3$ **then**
4: `base[`i`]` $= 3$; `hbt1[`i`]` $=$ `hbt2[`i`]` $= 0$; $k_1 = k_1/3$; $k_2 = k_2/3$;

The hybrid binary-ternary joint sparse form

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

Input: $k_1, k_2 > 0$

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

1: $i = 0$
2: **while** $k_1 > 0$ or $k_2 > 0$ **do**
3:     **if** $k_1 \equiv 0 \pmod 3$ and $k_2 \equiv 0 \pmod 3$ **then**
4:         `base[`i`]` $= 3$; `hbt1[`i`]` $=$ `hbt2[`i`]` $= 0$; $k_1 = k_1/3$; $k_2 = k_2/3$;
5:     **else if** $k_1 \equiv 0 \pmod 2$ and $k_2 \equiv 0 \pmod 2$ **then**
6:         `base[`i`]` $= 2$; `hbt1[`i`]` $=$ `hbt2[`i`]` $= 0$; $k_1 = k_1/2$; $k_2 = k_2/2$;

The hybrid binary-ternary joint sparse form

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

Input: $k_1, k_2 > 0$
Output: `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**
    $\texttt{base}[i] = 3$; $\texttt{hbt1}[i] = \texttt{hbt2}[i] = 0$; $k_1 = k_1/3$; $k_2 = k_2/3$;
  else if $k_1 \equiv 0 \pmod 2$ and $k_2 \equiv 0 \pmod 2$ **then**
    $\texttt{base}[i] = 2$; $\texttt{hbt1}[i] = \texttt{hbt2}[i] = 0$; $k_1 = k_1/2$; $k_2 = k_2/2$;
  else
    $\texttt{base}[i] = 2$; $\texttt{hbt1}[i] = k_1$ mods 6; $\texttt{hbt2}[i] = k_2$ mods 6;
    $k_1 = (k_1 - \texttt{hbt1}[i])/2$; $k_2 = (k_2 - \texttt{hbt2}[i])/2$;
  $i = i + 1$
return `hbt1[]`, `hbt2[]`, `base[]`

Theoretical analysis

It is possible to analyze the algorithm by means of Markov chains. We obtain the following probabilities:

$$p_2 = \frac{32}{59}, \quad p_3 = \frac{27}{59}, \quad p_z = \frac{35}{59}, \quad p_{nz} = \frac{24}{59}.$$

Now, using p_2 and p_3, we can evaluate the average base $\beta = 2.4078$ and deduce the average number of columns

$$(\log_\beta 2) \times n \approx 0.7888n.$$

Finally, we get that the expected number of elliptic curve additions per bit is approximately

$$\frac{24}{59} \times 0.7888 \approx 0.3209.$$

Comparisons

Theoretical comparison of HBTJSF, JSF and interleaving w-NAF for a n-bit pair of integers.

Parameters	HBTJF	JSF	Interleaving w-NAF
Average base	2.41	2	2
Avg # base 2 col.	$0.43n$	$n+1$	$n+1$
Avg # base 3 col.	$0.36n$	0	0
Avg # non-zero col.	$0.32n$	$0.5n$	$2n/(w+1)$
Precomp.	14	2	$2^{w-1}-2$

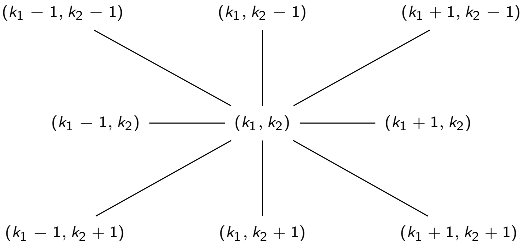
Implementation results confirm the advantage of db-chains for curves with fast tripling such as DIK3 curves.

	HBTJF	Inter 5-NAF	Inter. 4-NAF	JSF
163-bit	2065443	2207935	2303874	2407781
Improvement (%)	-	6.90	11.54	14.22
233-bit	3503081	3897876	3974763	4247352
Improvement (%)	-	11.27	13.46	17.52
571-bit	19608811	22303921	23084231	24656049
Improvement (%)	-	13.74	17.72	20.47

(Time in μs for 1000 experiments)

The tree-based approach

Given (k_1, k_2), consider the pairs $(k_1 + i, k_2 + j)$ with $i, j \in \{-1, 0, 1\}$.



Idea: Clear common powers of 2 and 3 from each pair and reapply. Not practical!

Only keep the branch with the largest common power of the form $2^a 3^b$.

Precomputations: $P, Q, P + Q, P - Q$

Complexity: Average joint density < 0.3945

Conclusions

Our last results on optimal db-chains suggest that there is still room for improvements.

Open problem: length of an optimal **signed** double-base chain.

Finding (hyper)elliptic curves with fast tripling does make sense.

Other group-like structures with fast cubing operation can benefit from those results. We are currently working on cubing over real quadratic fields.

Conclusions

Our last results on optimal db-chains suggest that there is still room for improvements.

Open problem: length of an optimal **signed** double-base chain.

Finding (hyper)elliptic curves with fast tripling does make sense.

Other group-like structures with fast cubing operation can benefit from those results. We are currently working on cubing over real quadratic fields.

Thanks!

Laurent.Imbert@ucalgary.ca

www.math.ucalgary.ca/~imbertl