Searching for Differential Addition Chains

Daniel J. Bernstein, **Jolijn Cottaar**, Tanja Lange

ANTS XVI, 2024

Introduction

Let P be a point on an elliptic curve and n a positive integer.
Goal: Compute nP.

Normally we compute by double&add, for example for $n = 29$:

$$P \to 2P \to 3P \to 6P \to 7P \to 14P \to 28P \to 29P.$$

As an addition chain:

$$1, 2, 3, 6, 7, 14, 28, 29.$$

These chains are used in elliptic-curve cryptography and isogeny-based cryptography.

Introduction

An *addition chain* for an integer n is defined as a sequence of integers

$$1 = c_0, c_1, \ldots, c_r = n$$

such that, for each $i \in \{1, \ldots, r\}$, there exist $j, k \in \{0, \ldots, i-1\}$ such that $c_i = c_j + c_k$.

An example for $n = 29$:

$$1, 2, 3, 6, 7, 14, 28, 29.$$

Montgomery showed on Montgomery curves (of the form $By = x^3 + Ax^2 + x$) that $x((a+b)P)$ takes just 6 field multiplications (5 if P affine), given $x(aP)$, $x(bP)$ and $x((a-b)P)$, instead of 8 field multiplications for regular point addition.

Introduction

A *differential addition chain* for an integer n is defined as a sequence of integers

$$1 = c_0, c_1, \ldots, c_r = n$$

such that, for each $i \in \{1, \ldots, r\}$, there exist $j, k \in \{0, \ldots, i-1\}$ such that $c_i = c_j + c_k$ and $c_j - c_k \in \{0, c_0, c_1, \ldots, c_{i-1}\}$.

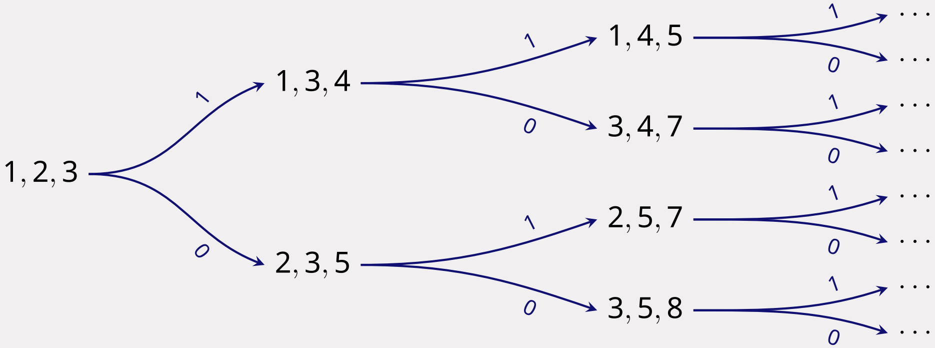
An example for $n = 29$:

$$1, 2, 3, 5, 8, 13, 16, 29$$

Saving of all intermediate results gives a $\Theta(\log(n))$ memory requirement.
Want 3-tuple of integers $(c - b, b, c)$, no inputs permitted from outside.

Introduction

Continued-fraction tuples are formed as follows:



Introduction

A sequence of tuples ending at $n = 29$:

$$(1,2,3),(2,3,5),(3,5,8),(5,8,13),(8,13,21),(8,21,29)$$

Continued-fraction differential addition chain:

$$1,2,3,5,8,13,21,29$$

Which is encoded as 00001.

Existing algorithms: CFRC, SIBC and CTIDH

Existing algorithms to find continued-fraction differential addition chains:

- Montgomery's CFRC algorithm [Mon92]: based on Euclid's algorithm.
- SIBC [ACDR21]: Brute force search in the tuples tree.
  - Tests all chain up to conjectured upper bound: $2 + \lfloor 1.5 \log_2(n) \rfloor$.
  - Overshooting gets pruned.
- CTIDH [BBC+21]: Also a brute force search in the tuples tree.
  - Increments the length of the chains considered.
  - Chains that do not reach the target when doubling get pruned.

Pruning algorithm

We propose an improved version of the approach in CTIDH:

- Incrementing lengths being considered.
- Chains that overshoot the target are pruned.
- Chains that undershoot are pruned using the Fibonacci bound.

Pruning algorithm

Fibonacci upper bound:
A chain of length i, with tuple (a_i, b_i, c_i):

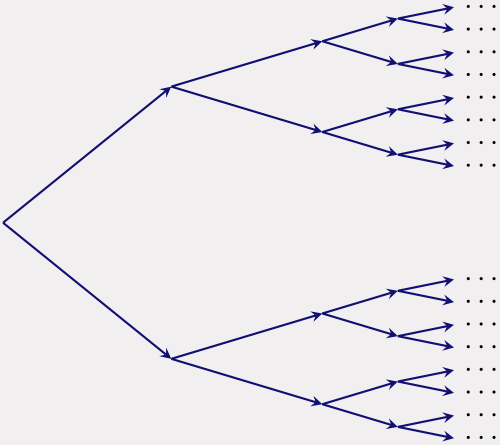
- Length $i+1$ will reach at most $2c_i$,
- length $i+2$ will reach at most $3c_i$,
- length $i+3$ will reach at most $5c_i$, etc.

I.e. when aiming for length r:

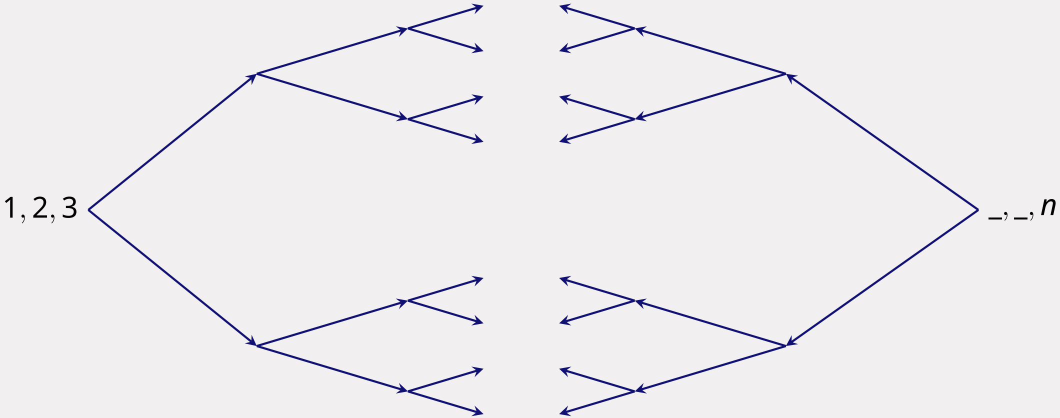
- $c_r \leq n-1$ will undershoot,
- $c_{r-1} \leq \lfloor (n-1)/2 \rfloor$ will undershoot,
- $c_{r-2} \leq \lfloor (n-1)/3 \rfloor$ will undershoot, etc.

Meet-in-the-middle algorithm

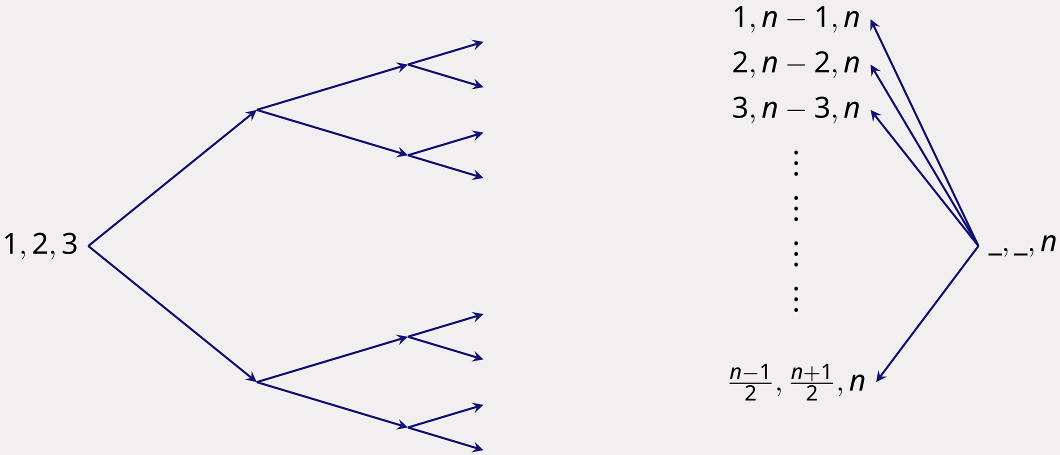
$1, 2, 3$



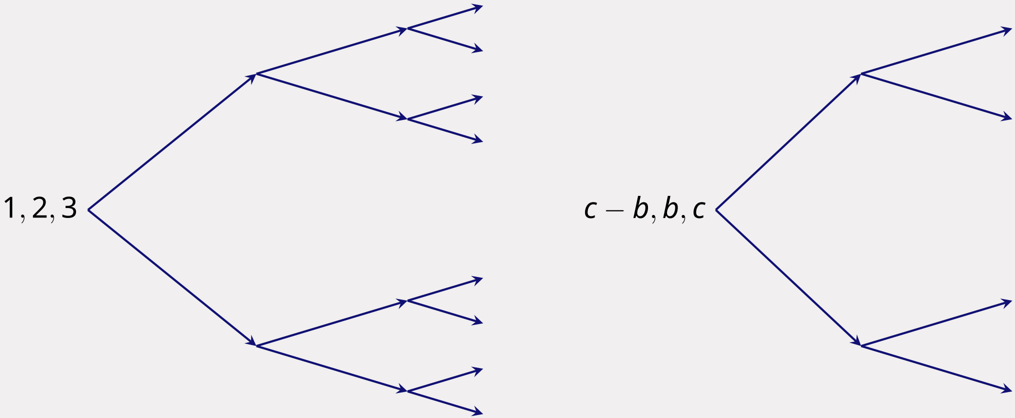
Meet-in-the-middle algorithm



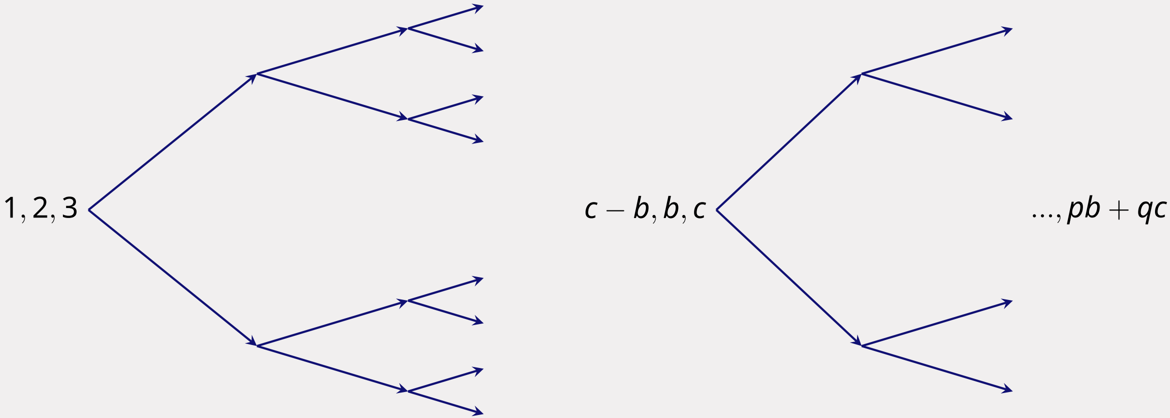
Meet-in-the-middle algorithm



Meet-in-the-middle algorithm



Meet-in-the-middle algorithm



Meet-in-the-middle algorithm

target → Left side → Right side → Mod comparisons → Final checks → chain

Meet-in-the-middle algorithm

$n = 29 \rightarrow$ Left side $\rightarrow$ Right side $\rightarrow$ Mod comparisons $\rightarrow$ Final checks $\rightarrow$ chain

Meet-in-the-middle algorithm

$n = 29$ → Left side → Right side → Mod comparisons → Final checks → chain

Create a list of continued-fraction tuples that adhere to:

- left-interval variant: final entry is in a certain interval based on the target.
- left-length variant: chains up to a certain length.

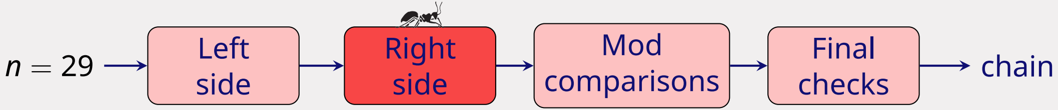
We find the following chains (for left-interval variant):

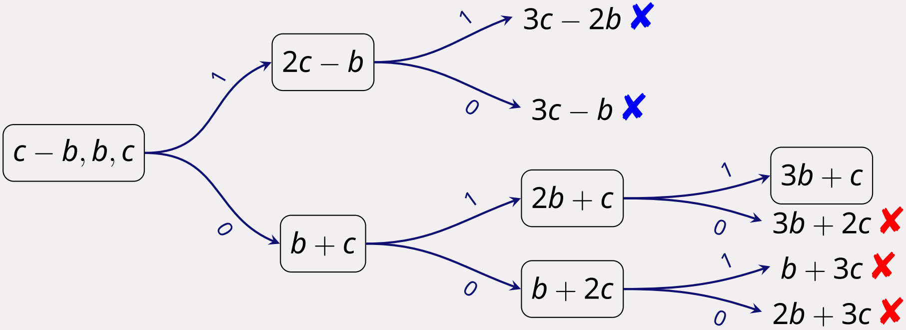
$$1, 2, 3, 5, 8, 11 \qquad 1, 2, 3, 4, 7, 10$$

$$1, 2, 3, 5, 7, 9, 11 \qquad 1, 2, 3, 4, 5, 6, 11$$

$$1, 2, 3, 4, 7, 11$$

Meet-in-the-middle algorithm





Meet-in-the-middle algorithm

$n = 29$ → Left side → Right side → **Mod comparisons** → Final checks → chain

Let $m = \lfloor n^{1/3} \rfloor = 3$. We create the following dictionary with $(b \mod m, c \mod m)$:

$$
\begin{aligned}
(0,2) &: 1,2,3,5,7,9,11; 1,2,3,4,5,6,11 \\
(1,1) &: 1,2,3,4,7,10 \\
(1,2) &: 1,2,3,4,7,11 \\
(2,2) &: 1,2,3,5,8,11
\end{aligned}
$$

Meet-in-the-middle algorithm

$n = 29$ → Left side → Right side → **Mod comparisons** → Final checks → chain

Now we test our found right-side chains to check if

$$(p \cdot b \mod m) + (q \cdot c \mod m) = n \mod m.$$

Meet-in-the-middle algorithm

$n = 29$ → Left side → Right side → **Mod comparisons** → Final checks → chain

Example with the chain $b + c$ (so $p = 1$, $q = 1$) for $29 \mod 3 = 2$:
We run over possible b and check if there is a $(b \mod 3, c \mod 3)$ which hold:

$$
\begin{array}{llllc}
b = 0 \mod 3 & \rightarrow 1 \cdot 0 + 1 \cdot (c \mod 3) = 2 \mod 3 & \rightarrow c = 2 \mod 3 & \rightarrow (0, 2) & \checkmark \\
b = 1 \mod 3 & \rightarrow 1 \cdot 1 + 1 \cdot (c \mod 3) = 2 \mod 3 & \rightarrow c = 1 \mod 3 & \rightarrow (1, 1) & \checkmark \\
b = 2 \mod 3 & \rightarrow 1 \cdot 2 + 1 \cdot (c \mod 3) = 2 \mod 3 & \rightarrow c = 0 \mod 3 & \rightarrow (2, 0) &
\end{array}
$$

Meet-in-the-middle algorithm

$n = 29$ → Left side → Right side → Mod comparisons → Final checks → chain

In total we get:

$$\ldots, c : \ldots, 9, 11; \ldots, 6, 11; \ldots, 7, 11; \ldots, 8, 11$$
$$\ldots, b + c : \ldots, 9, 11; \ldots, 6, 11; \ldots, 7, 10$$
$$\ldots, b + 2c : \ldots, 7, 11$$
$$\ldots, 2b + c : \ldots, 9, 11; \ldots, 6, 11$$
$$\ldots, 3b + c : \ldots, 9, 11; \ldots, 6, 11; \ldots, 7, 11; \ldots, 8, 11$$
$$\ldots, 2c - b : \ldots, 8, 11$$

Meet-in-the-middle algorithm

$n = 29$ → Left side → Right side → Mod comparisons → **Final checks** → chain

Test our found right-side chains to check if:

$$p \cdot b + q \cdot c = n$$

Thus only having to do 15 full checks instead of 30 orginally.

Meet-in-the-middle algorithm

$n = 29$ → Left side → Right side → Mod comparisons → Final checks → chain

$1, 2, 3, 4, 7, 11$ and $c - b, b, c, b + c, b + 2c$ combine to the continued-fraction chain

$$1, 2, 3, 4, 7, 11, 18, 29.$$

$1, 2, 3, 5, 7, 9, 11$ and $c - b, b, c, b + c, 2b + c$ also reach 29 but the continued-fraction chain

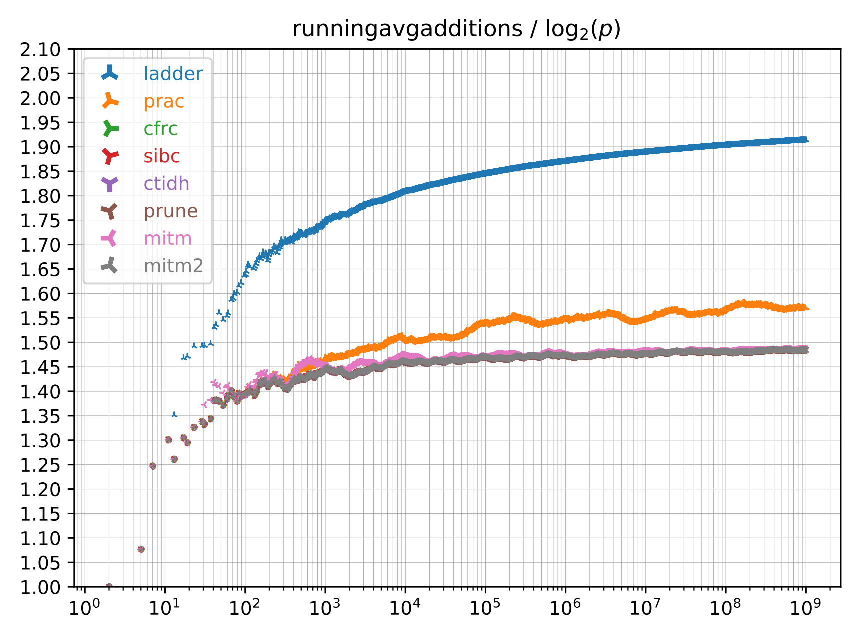
$$1, 2, 3, 5, 7, 9, 11, 20, 29$$

is one step longer.

Other considered chains

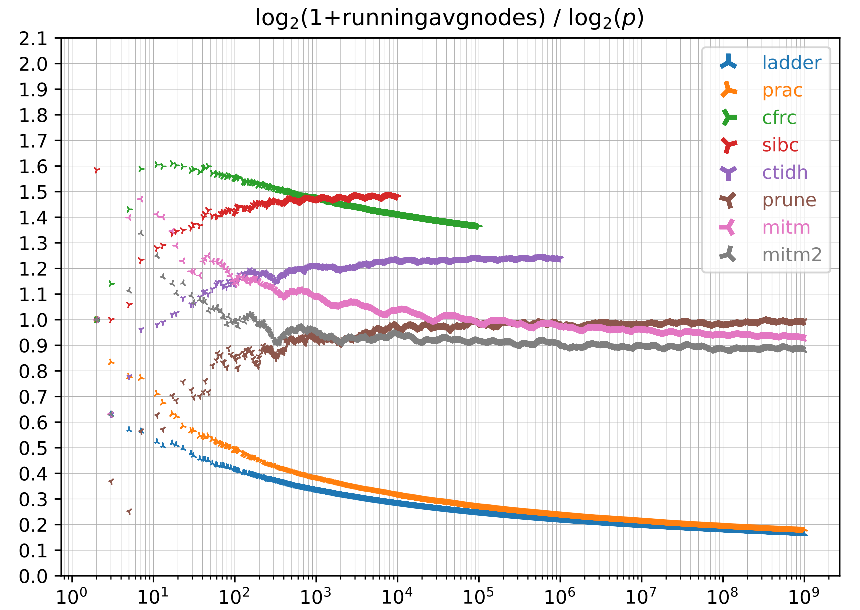
1. 'ladder': Standard ladder
2. 'prac': Montgomery's PRAC algorithm [Mon92] for all ρ, this generates a differential addition-subtraction chain (usually not a continued-fraction differential addition chain).

Results



runningavgadditions / log2(p)
ladder
prac
cfrc
sibc
ctidh
prune
mitm
mitm2
2.10
2.05
2.00
1.95
1.90
1.85
1.80
1.75
1.70
1.65
1.60
1.55
1.50
1.45
1.40
1.35
1.30
1.25
1.20
1.15
1.10
1.05
1.00
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9

Results



log2(1+runningavgnodes) / log2(p)
ladder
prac
cfrc
sibc
ctidh
prune
mitm
mitm2
2.1
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9

References I

Gora Adj, Jesús-Javier Chi-Domínguez, and Francisco Rodríguez-Henríquez.
SIBC Python library.
`https://github.com/JJChiDguez/sibc/`, 2021.

Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková.
CTIDH: faster constant-time CSIDH.
IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(4):351–387, 2021.
`https://tches.iacr.org/index.php/TCHES/article/view/9069`.

Peter L. Montgomery.
Evaluating recurrences of form $x_{m+n} = f(x_m, x_n, x_{m-n})$ via Lucas chains, 1992.