Faster cofactorization with ECM using mixed representations

Cyril Bouvier Laurent Imbert

LIRMM, CNRS, Univ. Montpellier, France

Séminaire CARAMBA – November 29th, 2018

Context

The Elliptic Curve Method (ECM) is the fastest known method for finding medium-size prime factors of large integers.

ECM is used as a subroutine of the Number Field Sieve (NFS), the most efficient algorithm for factoring integers of the form $N = pq$ with $p, q \approx \sqrt{N}$.
Also true for all NFS variants for computing discrete logarithms over finite fields.

ECM is used in the sieving phase of NFS (and descent for discrete log) during the cofactorization step; used to factor from millions to billions of integer of a hundred-ish bits.
RSA-768: cofactorization $\simeq 1/3$ of the sieving phase
$\simeq 5\,\%$ to $20\,\%$ of the total time

Goal

Speed up ECM in the context of the cofactorization step of NFS

Preliminaries

Scalar multiplication in stage 1 of ECM

Combination of blocks for stage 1 of ECM

Results and comparisons

Elliptic Curve Method (ECM)

Described by H. Lenstra in 1985; based on the ideas of $P-1$ algorithm.

ECM [in the case of projective Weierstrass curves]

Input: an integer N such that $\gcd(N,6)=1$ and a bound B_1
Output: a proper factor of N or FAILURE.

1: Choose an elliptic curve E over $\mathbb{Q}$ and a point $P \in E(\mathbb{Q})$
2: $k \leftarrow \operatorname{lcm}(2,3,4,\ldots,B_1) = \prod_{p \text{ prime} \leq B_1} p^{\lfloor \log(B_1)/\log(p) \rfloor}$
3: $Q \leftarrow [k]P$ ▷ computation done modulo N
4: **if** $1 < \gcd(Z_Q, N) < N$ **then** ▷ $Z_Q = Z$-coordinate of Q
5: **return** $\gcd(Z_Q, N)$
6: **else**
7: **return** FAILURE

Remark: the coordinate in the gcd can be different for other models of curves.

Some remarks on ECM

When does it succeed ? Let p be a prime factor of N

$$\begin{aligned} \#E(\mathbb{F}_p) \text{ is } B_1\text{-powersmooth} &\Rightarrow \text{the order of } P \text{ over } E(\mathbb{F}_p) \text{ is } B_1\text{-powersmooth} \\ &\Rightarrow Q = [k]P \text{ is the point at infinity on } E(\mathbb{F}_p) \\ &\Rightarrow Z_Q \equiv 0 \pmod{p} \\ &\Rightarrow p \mid \gcd(Z_Q, N) \end{aligned}$$

If ECM fails, we can try another curve and hope that the new group order will be B_1-powersmooth.

Cost of ECM: cost of the scalar multiplication $[k]P$

The model of the curves and the system of coordinates can be chosen; it influences

- the way the scalar multiplication is perform;
- the smoothness probability.

Stage 2 of ECM

As for similar algorithms, it exists a Stage 2 that is used to catch factor for which the group order fail to be B_1-powersmooth by just one prime larger the B_1.
Does not change ECM complexity but huge improvement in practice.

ECM – Stage 2 [in the case of projective Weierstrass curves]

Input: same as for Stage 1 + the point $Q = [k]P$ and a bound $B_2 \geq B_1$
Output: a proper factor of N or FAILURE.

1: **for all** primes $B_1 < \pi \leq B_2$ **do**
2: $R \leftarrow [\pi]Q$ ▷ computation done modulo N
3: **if** $1 < \gcd(Z_R, N) < N$ **then**
4: **return** $\gcd(Z_R, N)$
5: **return** FAILURE

Some variants reduce the number of gcd computed and perform the scalar multiplication more efficiently:

- Baby-step giant-step variant;
- FFT variant (useful for large value of B_2).

Elliptic cost and arithmetic cost

We want to compare the "cost" of a scalar multiplication.

Elliptic cost: number of elliptic operations (addition, doubling, tripling).

Arithmetic cost: number of arithmetic operations modulo N.
Only considered multiplications (**M**) and squarings (**S**).

To ease the comparisons, we assume $1\mathbf{S} = 1\mathbf{M}$
Assumption supported by experiments with CADO-NFS modular arithmetic functions for 64-bit, 96-bit and 128-bit integers.

Montgomery curves

Introduced by Montgomery in 1987 to speed up ECM.

Montgomery curve: let A and B such that $B(A^2 - 4) \neq 0$

$$E^M_{A,B} : BY^2Z = X^3 + AX^2Z + XZ^2.$$

XZ coordinate system: drop the Y coordinate.
Consequence: can only perform differential addition, *i.e.*, the sum of two points can be computed only if their difference is known.

Pros and cons: very fast elliptic operations but the use of a differential addition is a burden for the scalar multiplication algorithms.

Elliptic Operation	Notation	Input	$\rightarrow$	Output	Cost
Differential Addition	dADD	XZ	$\rightarrow$	XZ	$4\mathbf{M} + 2\mathbf{S}$
Doubling	dDBL	XZ	$\rightarrow$	XZ	$3\mathbf{M} + 2\mathbf{S}$

Edwards curves

Introduced by Edwards in 2007, considered for ECM by Bernstein *et al.* in 2010.
Twisted Edwards curve: let a and d such that $ad(a-d) \neq 0$

$$E^E_{a,d} : aX^2Z^2 + Y^2Z^2 = Z^4 + dX^2Y^2.$$

Two other coordinate systems are used for efficiency: completed and extended.
Twisted Edwards curves have a efficient point tripling.
We only consider twisted Edwards curves with $a = -1$: better, faster.

Elliptic Operation	Notation	Input	$\rightarrow$	Output	Cost
Addition	$\mathrm{ADD}_{\mathrm{comp}}$	ext.	$\rightarrow$	comp.	$4\mathbf{M}$
	ADD	ext.	$\rightarrow$	proj.	$7\mathbf{M}$
	$\mathrm{ADD}_{\varepsilon}$	ext.	$\rightarrow$	ext.	$8\mathbf{M}$
Doubling	DBL	ext. or proj.	$\rightarrow$	proj.	$3\mathbf{M} + 4\mathbf{S}$
	$\mathrm{DBL}_{\varepsilon}$	ext. or proj.	$\rightarrow$	ext.	$4\mathbf{M} + 4\mathbf{S}$
Tripling	TPL	ext. or proj.	$\rightarrow$	proj.	$9\mathbf{M} + 3\mathbf{S}$
	$\mathrm{TPL}_{\varepsilon}$	ext. or proj.	$\rightarrow$	ext.	$11\mathbf{M} + 3\mathbf{S}$

The best of both worlds

Better ? Montgomery or Edwards ? Depends on B_1 and the algorithm used for the scalar multiplication.

Every twisted Edwards curve is birationally equivalent to a Montgomery curve with

$$A = 2(a+d)(a-d) \text{ and } B = 4/(a-d).$$

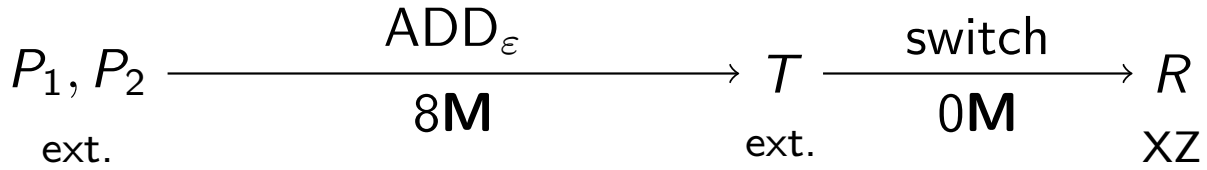
We will use this equivalence in the scalar multiplication of ECM:

- start the computation on a twisted Edwards curve;
- switch to the equivalent Montgomery curve;
- finish the computation on the Montgomery curve.

This equivalence was used to speed up the doubling in a YZ coordinate system on Edwards curves and, more recently, in the SIDH context.

Add and switch

The switch from twisted Edwards to Montgomery is always done after an addition.



Add and switch

The switch from twisted Edwards to Montgomery is always done after an addition.

$$\underset{\text{ext.}}{P_1, P_2} \xrightarrow[4\mathbf{M}]{\mathrm{ADD}_{\text{comp}}} \underset{\text{comp.}}{T'} \xrightarrow{4\mathbf{M}} \underset{\text{ext.}}{T} \xrightarrow[0\mathbf{M}]{\text{switch}} \underset{\text{XZ}}{R}$$

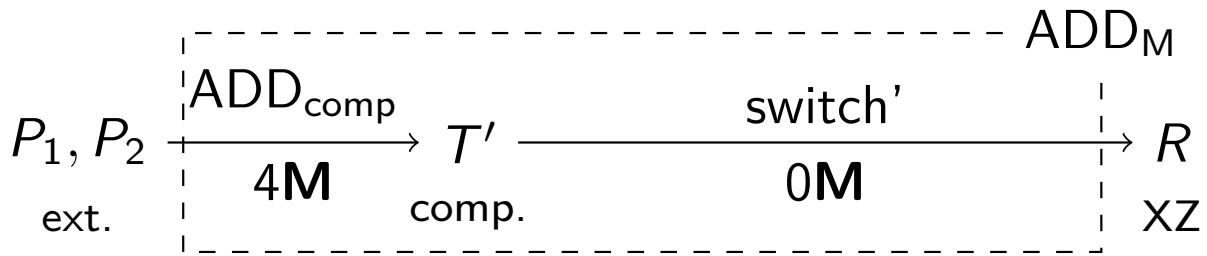
Add and switch

The switch from twisted Edwards to Montgomery is always done after an addition.

$$\underset{\text{ext.}}{P_1, P_2} \xrightarrow[4\mathbf{M}]{\mathrm{ADD}_{\text{comp}}} \underset{\text{comp.}}{T'} \xrightarrow[0\mathbf{M}]{\text{switch'}} \underset{\text{XZ}}{R}$$

Add and switch

The switch from twisted Edwards to Montgomery is always done after an addition.



Elliptic Operation	Notation	Input	$\rightarrow$	Output	Cost
Add & Switch	$\mathrm{ADD_M}$	Edwards ext.	$\rightarrow$	Montgomery XZ	4**M**

Remark: this elliptic operation is not "invertible" as the Y coordinate on the Montgomery is not computed.

ECM in the cofactorization step

During the cofactorization step of NFS (and its variants), ECM is used

- with small values of B_1 and B_2.
  Example of values used in CADO-NFS: $B_1 = 115$ and $B_2 = 5775$, $B_1 = 260$ and $B_2 = 12915$, $B_1 = 840$ and $B_2 = 42105$, ...

- with values of B_1 and B_2 known in advance.

Goal

Use precomputation to find the more efficient way to perform the scalar multiplication of stage 1 of ECM for the values of B_1 used during the cofactorization step

Preliminaries

Scalar multiplication in stage 1 of ECM

Combination of blocks for stage 1 of ECM

Results and comparisons

A particular scalar multiplication

Recall that stage 1 of ECM consists of multiplying a point P by the scalar

$$k = \prod_{p \text{ prime} \leq B_1} p^{\lfloor \log_p(B_1) \rfloor}$$

The best way to compute this scalar multiplication depends on B_1 and on the model of elliptic curves used.

Traditional scalar mulitplication algorithms use binary representation of the scalar. For example, double-and-add uses an unsigned representation, NAF uses a signed one. In those cases:

- #elliptic doublings = length of the representation - 1
- #elliptic additions = Hamming weight (= number of non-zero digits) - 1

Dixon and Lenstra's idea

$$k = \prod_{p \text{ prime} \leq B_1} p^{\lfloor \log_p(B_1) \rfloor}$$

Two naive possibilities to compute $[k]P$:

- compute k and perform one scalar multiplication by k;
- perform, for each prime $p \leq B_1$, exactly $\lfloor \log_p(B_1) \rfloor$ scalar multiplications by p.

Dixon and Lenstra's idea: gather prime factors of k in blocks such that the product of primes in a block has low Hamming weight.

Example: Let $p_1 = 1028107$, $p_2 = 1030639$ and $p_3 = 1097101$.

- p_1, p_2 and p_3 have respective Hamming weights 10, 16 and 11.
- The Hamming weight of the product $p_1 p_2 p_3$ is 8.

Bos and Kleinjung's improvement

Unlike Dixon and Lenstra, Bos and Kleinjung considered NAF representations, *i.e.*, *signed* binary representation.

Dixon and Lenstra considered all blocks with at most 3 primes
Bos and Kleinjung generated blocks with more primes.

- Cannot compute all possible blocks anymore (more than 2^{36} for $B_1 = 128$)
- Use the opposite strategy: they generate a huge quantity of integers with very low Hamming weights in NAF form and check if they correspond to valid blocks (using smoothness tests).

  Example: Let $B_1 = 32$:
  - $1000000000100001_2 = 2^{16} + 2^5 + 1 = 7 \times 17 \times 19 \times 29$ ✓
  - $1000000000010001_2 = 2^{16} + 2^4 + 1 = 3 \times 21851$ ✗

Computation of blocks

We consider other algorithms to compute the scalar multiplications:

- for the part on the twisted Edwards model:
  - double-base expansions
  - double-base chains
- for the part on the Montgomery model:
  - Lucas chains

Following Bos and Kleinjung's approch,

- we generate efficient chains/expansions
- and then check if they correspond to a valid block.

Double-base expansions

Let n be a positive integer, a double-base expansion for n is a way of writing n as

$$n = \sum_{i=0}^{m} \pm 2^{d_i} 3^{t_i},$$

where $|2^{d_i} 3^{t_i}| > |2^{d_j} 3^{t_j}|$ for all $0 \leq i < j \leq m$.

Given a double-base expansion for n, one can compute $[n]P$ with

- $D = \max_i d_i$ doublings,
- $T = \max_i t_i$ triplings,
- at most m additions,
- and at most $m + 1$ additional points for storing intermediate computation.

Double-base expansions

To avoid useless redundancies,

- we only generate double-base expansions whose terms have no common factors;
- we avoid generating double-base expansions for both n and $-n$ by imposing that $\pm 2^{d_0} 3^{d_0}$ be positive.

We generated double-base expansions

- with $m \in \{1, 2, 3\}$, *i.e.*, a small number of additions,
- and a large number of doublings and/or triplings.

We generated $3.04 \cdot 10^{12}$ double-base expansions; around $9 \cdot 10^7$ of them correspond to a valid block for $B_1 = 2^{13}$.

To go beyond $m = 3$ and in order to generate more integers of potential interest, we considered a subset of double-base expansions.

Double-base chains

A double-base chain for n is a double-base expansion

$$n = \sum_{i=0}^{m} \pm 2^{d_i} 3^{t_i},$$

where $|2^{d_j} 3^{t_j}|$ divides $|2^{d_i} 3^{t_i}|$ for all $0 \leq i < j \leq m$.

With an evaluation *à la Horner*, one can compute $[n]P$ with

- $D = d_0$ doublings,
- $T = t_0$ triplings,
- exactly m additions,
- and no additional storage.

Double-base chains

To avoid useless redundancies,

- we only generate double-base chains whose terms have no common factors, *i.e.*, with $\pm 2^{d_m} 3^{t_m} = \pm 1$;
- we only generate double-base chains for positive n by imposing that $\pm 2^{d_0} 3^{d_0}$ be positive.

We generated double-base expansions

- with $m \in [1, 8]$, *i.e.*, a small number of additions,
- and a large number of doublings and/or triplings.

We generated $2.57 \cdot 10^{13}$ double-base chains; around $2 \cdot 10^9$ of them correspond to a valid block for $B_1 = 2^{13}$.

Note: NAF representation is a double-base chain with $T = 0$.

Lucas chains

The two previous constructions does not work on Montgomery curves as they rely on an addition.
For Montgomery curves, we generated Lucas chains.

Let n be a positive integer. A Lucas chain of length ℓ for n is a sequence of integers $(c_0, c_1, \ldots, c_\ell)$ such that

- $c_0 = 1$,
- $c_\ell = n$,
- and for every $1 \leq i \leq \ell$,
  - either it exists $j < i$ such that $c_i = 2c_j$
  - or there exist $j_0, j_1, j_d < i$ such that $c_i = c_{j_0} + c_{j_1}$ and $c_{j_d} = \pm(c_{j_0} - c_{j_1})$.

In general, Lucas chains require more elliptic operations than binary, NAF, or double-base chains.

PRAC algorithm

PRAC: efficient algorithm by Montgomery to generate Lucas chains.

Sketch of the algorithm for computing $[n]P$:

- Start with $A = [2]P$, $B = C = P$ and $d = n - \lfloor n/\phi \rceil$ and $e = 2\lfloor n/\phi \rceil - n$.
- Invariants: $\pm C = A - B$ and $[n]P = [d]A + [e]B$.
- At each step, one rule (out of 9) is chosen based on the values of d and e.
- Stop when $d = e = 1$.

The corresponding Lucas chains is obtained from the successive values of d and e.

To generate Lucas chains, we reversed the algorithm:

- generate all possible combinations of rules of a given length (in practice, up to 13);
- compute the correspondings integers and check if they correspond to valid blocks.

Around $5 \cdot 10^6$ Lucas chains generated for $B_1 = 2^{13}$.

Preliminaries

Scalar multiplication in stage 1 of ECM

Combination of blocks for stage 1 of ECM

Results and comparisons

Notations

Let $\mathcal{B}$ be the set of all generated blocks. For each block $b \in \mathcal{B}$, we define

- $n(b)$: the integer associated to b;
- $\mathcal{M}_b$: the multiset composed of the prime factors (counted with multiplicity) of $n(b)$;
- $\mathrm{cost}(b)$: the arithmetic cost of b, *i.e.*, the sum of the costs of the elliptic operations used to compute the scalar multiplication by $n(b)$ using the algorithm associated to b;
- $\mathrm{acpb}(b)$: the arithmetic cost per bit, *i.e.*, $\mathrm{cost}(b)/\log_2(n(b))$.

We use the same notations for a set of blocks $\mathcal{A} \subset \mathcal{B}$. The generalization to a set of blocks $\mathcal{A}$ is straightforward, except for the cost:

$$\mathrm{cost}(\mathcal{A}) = \sum_{b \in \mathcal{A}} \mathrm{cost}(b) + \delta(\mathcal{A}) \underbrace{(\mathrm{cost}(\mathrm{ADD_M}) - \mathrm{cost}(\mathrm{ADD}_\varepsilon))}_{-4\mathbf{M}},$$

where

$$\delta(\mathcal{A}) = \begin{cases} 1 \text{ if } \mathcal{A} \text{ contains at least 1 PRAC block} \\ 0 \text{ otherwise} \end{cases}$$

Goal

Let B_1 be the smoothness bound for ECM stage 1.

$\mathcal{M}_{B_1}$ is defined as the multiset composed of all primes p less than or equal to B_1, each occurring exactly $\lfloor \log_p(B_1) \rfloor$ times.
By definition, k is equal to $\prod_{p \in \mathcal{M}_{B_1}} p$.

Goal of the combination algorithm

Find a subset $\mathcal{S}$ of $\mathcal{B}$ such that $\bigcup_{b \in \mathcal{S}} \mathcal{M}_b = \mathcal{M}_{B_1}$, *i.e.*, $\prod_{b \in \mathcal{S}} n(b) = k$, which minimizes cost($\mathcal{S}$).

It reformulate Dixon and Lenstra's idea using our notation.

Bos and Kleinjung's algorithm

Bos and Kleinjung described a fast algorithm to compute a non-optimal solution:

- start with $\mathcal{M} = \mathcal{M}_{B_1}$ and $\mathcal{S} = \emptyset$;
- repeat until $\mathcal{M} \neq \emptyset$:
  - pick the "best" block $b \in \mathcal{B}$ such that $\mathcal{M}_b \subseteq \mathcal{M}$ and the ratio $\mathrm{dbl}(b)/\mathrm{add}(b)$ is large enough
  - add b in $\mathcal{S}$ and subtract $\mathcal{M}_b$ from $\mathcal{M}$.
- return $\mathcal{S}$.

At each iteration, the "best" block is chosen as the one that minimizes the following score function, defined when $\mathcal{M}_b \neq \emptyset$ and $\mathcal{M}_b \subseteq \mathcal{M}$:

$$\mathrm{score}(b, \mathcal{M}) = \sum_{\substack{\ell=1 \\ a_\ell(\mathcal{M})\neq 0}}^{\lceil \log_2(\max(\mathcal{M})) \rceil} \frac{a_\ell(\mathcal{M}_b)}{a_\ell(\mathcal{M})} \text{ with } a_\ell(\mathcal{M}) = \frac{\#\{p \in \mathcal{M} \mid \lceil \log_2(p) \rceil = \ell\}}{\#\mathcal{M}}.$$

This function is defined to favor blocks whose multisets share many large factors with the current multiset $\mathcal{M}$ of remaining factors.

They also presented a randomized version of the algorithm.

Ishii *et al.*'s algorithm

Ishii *et al.* kept the same algorithm and replaced the bound on the ratio $\mathrm{dbl}(b)/\mathrm{add}(b)$ by κ:

$$\kappa(b) = \frac{\log_2(n(b))}{\mathrm{dbl}(b) + 8/7\,\mathrm{add}(b) - \log_2(n(b))}.$$

It produces slightly better results than Bos and Kleinjung's algorithm.

The ratio $\mathrm{dbl}(b)/\mathrm{add}(b)$ and κ are not easily adaptable to our setting because

- we also use triplings;
- we use both twisted Edwards and Montgomery curves.

Notice that, on twisted Edwards curves, the function κ is closely related to the arithmetic cost per bit of a NAF block. Indeed, we have

$$\mathrm{acpb}(b) \simeq \frac{7\,\mathrm{dbl}(b) + 8\,\mathrm{add}(b)}{\log_2(n(b))} = \frac{7}{\kappa(b)} + 7.$$

The score function

For our combination algorithm, we decided not to use the score function from Bos and Kleinjung's algorithm.

We observed that it does not always achieve its goal to favor blocks with many large factors.

Example: Let $B_1 = 256$:

$\mathcal{M}_b$	$\text{score}(b, \mathcal{M}_{B_1})$
$\{233, 193, 163\}$	3.043
$\{233, 193, 179, 109, 103, 73\}$	4.214
$\mathcal{M}_{B_1}$	8

Remember that the algorithm choose the block with the smallest score.

Our algorithm

We were not able to find a suitable replacement for the score function.
And using only acpb to sort blocks did not yield better results.
Thus, we tried a more exhaustive approch.

A complete exhaustive search is totally out of reach, even for not-so-large values of B_1.

To reduce the enumeration depth, we impose the use of a bound on the number of blocks in the solution set $\mathcal{S}$.

To reduce the width of each step in the enumeration, we use the fact that we know an upper bound on the minimal cost.

Exploiting an upper bound on the minimal cost

Let C be an upper bound on the arithmetic cost of the best solution set.

Let $\mathcal{S}_0 \subseteq \mathcal{B}$ be a partial solution, i.e., such that $\bigcup_{b \in \mathcal{S}_0} \mathcal{M}_b \subsetneq \mathcal{M}_{B_1}$.
Then a solution set $\mathcal{S}$ containing $\mathcal{S}_0$ satisfies $\text{cost}(\mathcal{S}) < C$, only if $\mathcal{S} \setminus \mathcal{S}_0$ contains at least one block whose arithmetic cost per bit is not greater than

$$\text{acpb}_{\max} = \frac{C - (\text{cost}(\mathcal{S}_0) + (1 - \delta(\mathcal{S}_0))(\text{cost}(\text{ADD}_\text{M}) - \text{cost}(\text{ADD}_\varepsilon)))}{\log_2(n(\mathcal{M}_{B_1})) - \log_2(n(\mathcal{S}_0))}. \quad (1)$$

Our algorithm:

- Sort $\mathcal{B}$, the set of all generated blocks, by increasing value of acpb
- Enumerate all subsets of $\mathcal{B}$ of length less than a given bound ℓ
  - at each step of the enumeration, only consider blocks $b \in \mathcal{B}$ such that $\text{acpb}(b) \leq \text{acpb}_{\max}$
  - the bound C on the arithmetic cost of the best solution set can be updated during the algorithm

Preliminaries

Scalar multiplication in stage 1 of ECM

Combination of blocks for stage 1 of ECM

Results and comparisons

Cost comparison for stage 1

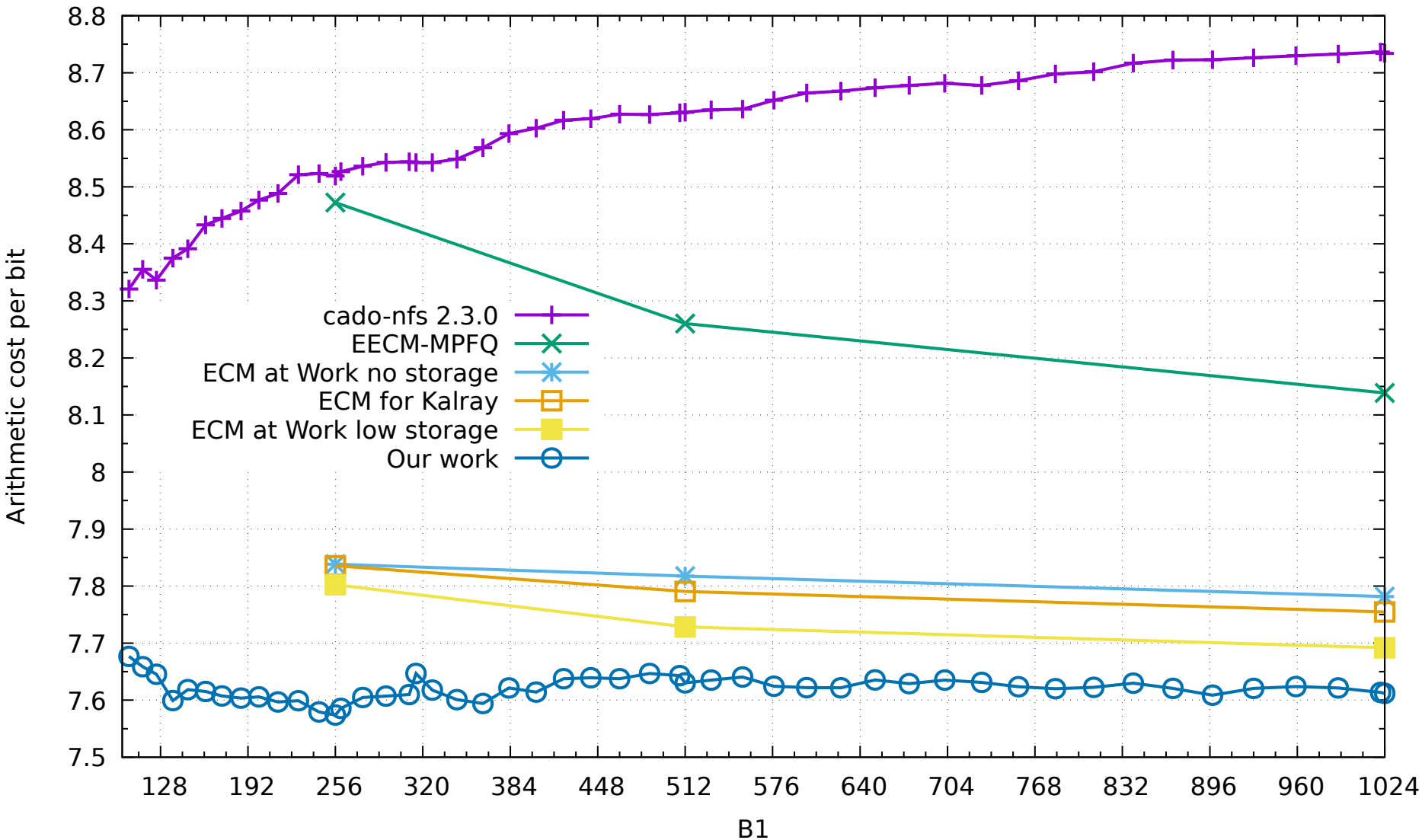
Every implementation in this comparison uses twisted Edwards curves, except CADO-NFS 2.3.0 which uses Montgomery curve and our work which uses a mix of both.

$B_1 =$	256	512	1024	8192
CADO-NFS 2.3.0	3091	6410	12916	104428
EECM-MPFQ	3074	6135	12036	93040
ECM at work[1] (no storage)	2844	5806	11508	91074
ECM on Kalray[2]	2843	5786	11468	90730
ECM at work[1] (low storage)	2831	5740	11375	89991
this work	2748	5667	11257	89572

Table: Number of modular multiplication (**M**) for various implementations of ECM (stage 1) and some commonly used smoothness bounds B_1, assuming $1\mathbf{S} = 1\mathbf{M}$

[1] Bos and Kleinjung
[2] Ishii *et al.*

Cost comparison for stage 1



Cost comparison for stage 2

In our implementation, stage 1 always outputs a point on a Montgomery curve. Is stage 2 faster for Montgomery curves or twisted Edwards curves ?

We use CADO-NFS implementation of stage 2 on Montgomery curves with a few changes.

	$B_1 =$	256	512	1024	8192
	$B_2 =$	2^{14}	$3 \cdot 2^{14}$	$7 \cdot 2^{14}$	$80 \cdot 2^{14}$
CADO-NFS 2.3.0		2387	6120	13264	134761
ECM on Kalray[1]		2538	5812	11410	91122
this work		2227	5160	10273	89866

Table: Number of modular multiplications (**M**) for ECM stage 2 assuming 1**S** = 1**M**

[1] same as ECM at Work for stage 2; it is based on Miele's thesis

Cost comparison for CADO-NFS default strategy

#	B_1	B_2	CADO-NFS 2.3.0			after this work		
			stage 1	stage 2	total	stage 1	stage 2	total
1	105	3255	1240	759	1999	1144	734	1878 (-6.1%)
2	315	5355	3900	1028	4928	3491	1011	4502 (-8.6%)
3	115	5775	1415	1071	2486	1297	1055	2352 (-5.4%)
4	125	6195	1460	1122	2582	1339	1104	2443 (-5.4%)
5	137	6825	1652	1209	2861	1499	1183	2682 (-6.3%)
10	200	9975	2521	1604	4125	2262	1566	3828 (-7.2%)
20	364	18165	4477	2591	7068	3968	2408	6376 (-9.8%)
30	577	28875	7174	3813	10987	6322	3409	9731 (-11.4%)

Table: Number of modular multiplications (**M**) for ECM used in the default strategy of CADO-NFS, assuming $1\mathbf{S} = 1\mathbf{M}$

Implementation in CADO-NFS

We added code to support usage of twisted Edwards curves:

- structure for an elliptic point that can be used for Montgomery curves, twisted Edwards curves (and even Weierstrass curves);
- functions implementing elliptic doublings and additions for all the coordinate systems necessary.

We updated the code for stage 2:

- remove hardcoded baby-step giant-step parameter; choose it according to B_1 and B_2;
- small improvement in the giant-step scalar multiplications;
- new function to build the set of baby-step values.

We added harcoded combinations for values of B_1 used in the default strategy:

- combinations are computed with our algorithm;
- hardcoded in a header in a "compressed" and easily-parsable format.

Conclusion

We proposed a improvement for ECM in the context of cofactorization step of NFS and its variants.

Following the works from Dixon and Lenstra and Bos and Kleinjung,

- we generated chains of various types;
- we combined them using a quasi exhaustive approach for various B_1-values used in the cofactorization step.

Our ECM implementation uses

- both twisted Edwards curves and Montgomery curves;
- a new addition-and-switch operation to go from one model to the other;
- uses double-base expansions and chains and PRAC-generated Lucas chains.

For $B_1 \leq 8192$, our implementation requires fewer modular multiplications than any other publicly available implementation of ECM.

Thank you for your attention !
Any questions ?

Bonus: stage 2 of ECM

Stage 2 of ECM

Recall stage 2 algorithm:

ECM – Stage 2 [in the case of projective Weierstrass curves]

Input: same as for Stage 1 + the point $Q = [k]P$ and a bound $B_2 \geq B_1$
Output: a proper factor of N or FAILURE.

1: **for all** primes $B_1 < \pi \leq B_2$ **do**
2: $R \leftarrow [\pi]Q$ ▷ computation done modulo N
3: **if** $1 < \gcd(Z_R, N) < N$ **then**
4: **return** $\gcd(Z_R, N)$
5: **return** FAILURE

In practice, for values of B_2 used in the cofactorization step, the baby-step giant-step variant is used.

Baby-step Giant-step for stage 2 of ECM

Let Q be the output point of stage 1, B_2 the stage 2 bound and ω a positive integer (coprime with all primes in $]B_1, B_2]$).
Two sets U and V are defined as

$$U = \{u \in \mathbb{Z} \mid 1 \leq u \leq \frac{\omega}{2}, \gcd(u, \omega) = 1\}$$

$$V = \{v \in \mathbb{Z} \mid \left\lceil \frac{B_1}{\omega} - \frac{1}{2} \right\rceil \leq v \leq \left\lfloor \frac{B_2}{\omega} - \frac{1}{2} \right\rfloor\}$$

Facts:

- every $B_1 < \pi \leq B_2$ can be written as $v\omega \pm u$, with $u \in U$ and $v \in V$.
- $[\pi]Q$ is the point at infinity if $\pm[u]Q$ and $[v\omega]Q$ are opposite points

New stage 2:

- compute $[u]Q$ for $u \in U$, $[\omega]Q$ and $[v\omega]Q$ for $v \in V$;
- compute $\gcd(m, N)$ where m is defined by (for Montgomery curves):

$$m = \prod_{u \in U} \prod_{v \in V} Z_{[u]Q} X_{[v\omega]Q} - Z_{[v\omega]Q} X_{[u]Q}$$

Stage 2 in CADO-NFS

In practice, to reduce the number of factors in the product to compute m, we use:

$$UV = \{(u, v) \in U \times V \mid v\omega \pm u \text{ is a prime in }]B_1, B_2]\}$$

$$m = \prod_{(u,v)\in UV} Z_{[u]Q} X_{[v\omega]Q} - Z_{[v\omega]Q} X_{[u]Q}$$

Example: with $B_1 = 256, B_2 = 16384, \omega = 210$

- $\#U \times \#V = 24 \times 77 = 1848$
- $\#UV = 1381$

First improvement: UV can be reduced.
Let (u, v) in UV, p and q two stage 2 primes such that $p = v\omega \pm u$ and q is a multiple of $v\omega \mp u$. Then, the pair corresponding to q can be removed from UV.

Example: with $B_1 = 256, B_2 = 16384, \omega = 210$

- in CADO-NFS 2.3.0: $\#UV = 1298$
- in CADO-NFS current master: $\#UV = 1294$

Stage 2 in CADO-NFS

Second improvement: homogenize the Z-coordinates of all points appearing in m.

$$X_u = X_{[u]Q} \times \prod_{\tilde{u} \in U \setminus \{u\}} Z_{[\tilde{u}]Q} \times \prod_{\tilde{v} \in V} Z_{[\tilde{v}\omega]Q} \qquad \text{for } u \in U$$

$$X_v = X_{[v\omega]Q} \times \prod_{\tilde{u} \in U} Z_{[\tilde{u}]Q} \times \prod_{\tilde{v} \in V \setminus \{v\}} Z_{[\tilde{v}\omega]Q} \qquad \text{for } v \in V$$

At the end, compute $\gcd(\tilde{m}, N)$ where

$$\tilde{m} = \prod_{(u,v) \in UV} X_u - X_v$$

Cost:

- $3\#UV$ multiplications to compute m
- $4(\#U + \#V) - 6 + \#UV$ multiplications to compute $\tilde{m}$

Example: with $B_1 = 256, B_2 = 16384, \omega = 210$

- $3 \times 1294 = 3882\mathbf{M}$ to compute m
- $4 \times (24 + 72) - 6 + 1294 = 378 + 1294 = 1672\mathbf{M}$ to compute $\tilde{m}$

New stage 2 in CADO-NFS

CADO-NFS 2.3.0 always used $\omega = 210$.

First improvement is used to reduce the size of the set UV.

Baby-step (assumes $6 \mid \omega$):

- compute $[2]Q$,$[3]Q$,$[5]Q$,$[6]Q$,$[7]Q$,$[11]Q$;
- compute $[5 + 6k]Q$ and $[1 + 6k]Q$ for $2 \leq k < \left\lfloor \frac{\omega}{6} \right\rfloor / 2$;
- compute $[\omega]Q$ using previously computed points.

Giant-step:

- compute $[\min V][\omega]Q$;
- compute $[\min V + 1][\omega]Q$;
- compute $[i][\omega]Q$ for $\min V + 2 \leq i \leq \max V$ with 1 dADD for each i.

The product is computed using the second improvement.

New stage 2 in CADO-NFS

CADO-NFS 2.3.0 always used $\omega = 210$.

First improvement is used to reduce the size of the set UV.
The set UV could still be slightly reduced.

Baby-step (assumes $6 \mid \omega$):

- compute $[2]Q$,$[3]Q$,$[5]Q$,$[6]Q$,$[7]Q$,$[11]Q$;
- compute $[5 + 6k]Q$ and $[1 + 6k]Q$ for $2 \leq k < \left\lfloor \frac{\omega}{6} \right\rfloor /2$;
- compute $[\omega]Q$ using previously computed points.

Giant-step:

- compute $[\min V][\omega]Q$;
- compute $[\min V + 1][\omega]Q$;
- compute $[i][\omega]Q$ for $\min V + 2 \leq i \leq \max V$ with 1 dADD for each i.

The product is computed using the second improvement.

New stage 2 in CADO-NFS

CADO-NFS 2.3.0 always used $\omega = 210$.

First improvement is used to reduce the size of the set UV.
The set UV could still be slightly reduced.

Baby-step (assumes $6 \mid \omega$):

- compute $[2]Q$,$[3]Q$,$[5]Q$,$[6]Q$,$[7]Q$,$[11]Q$;
- compute $[5+6k]Q$ and $[1+6k]Q$ for $2 \leq k < \left\lfloor \frac{\omega}{6} \right\rfloor /2$;
- compute $[\omega]Q$ using previously computed points.

Giant-step:

- ~~compute $[\min V][\omega]Q$;~~
- ~~compute $[\min V + 1][\omega]Q$;~~

  } compute both $[\min V][\omega]Q$ and $[\min V+1][\omega]Q$ with only one Montgomery ladder

- compute $[i][\omega]Q$ for $\min V + 2 \leq i \leq \max V$ ~~with 1 dADD for each i.~~ with 1 dDBL for even $i \geq 2 \min V$ and 1 dADD otherwise.

The product is computed using the second improvement.

New stage 2 in CADO-NFS

~~CADO-NFS 2.3.0 always used $\omega = 210$~~.
New code choose the best ω.

First improvement is used to reduce the size of the set UV.
The set UV could still be slightly reduced.

Baby-step (assumes $6 \mid \omega$):

- compute $[2]Q$,$[3]Q$,$[5]Q$,$[6]Q$,$[7]Q$,$[11]Q$;
- compute $[5+6k]Q$ and $[1+6k]Q$ for $2 \leq k < \left\lfloor \frac{\omega}{6} \right\rfloor /2$;
- compute $[\omega]Q$ using previously computed points.

Giant-step:

- ~~compute $[\min V][\omega]Q$;~~
- ~~compute $[\min V + 1][\omega]Q$;~~

  } compute both $[\min V][\omega]Q$ and $[\min V+1][\omega]Q$ with only one Montgomery ladder

- compute $[i][\omega]Q$ for $\min V + 2 \leq i \leq \max V$ ~~with 1 dADD for each i.~~ with 1 dDBL for even $i \geq 2 \min V$ and 1 dADD otherwise.

The product is computed using the second improvement.

New stage 2 in CADO-NFS

$B_1 = 256$ $B_2 = 2^{14}$	ω	Baby-step		Giant-step		product	total
		dDBL	dADD	dDBL	dADD	**M**	**M**	**M**
CADO-NFS 2.3.0	210	3	37	6	74	378	1298	2387

$B_1 = 1024$ $B_2 = 7 \cdot 2^{14}$	ω	Baby-step		Giant-step		product	total
		dDBL	dADD	dDBL	dADD	**M**	**M**	**M**
CADO-NFS 2.3.0	210	3	37	12	505	2078	7859	13264

New stage 2 in CADO-NFS

$B_1 = 256$ $B_2 = 2^{14}$	ω	Baby-step dDBL	Baby-step dADD	Giant-step dDBL	Giant-step dADD	product **M**	product **M**	total **M**
CADO-NFS 2.3.0	210	3	37	6	74	378	1298	2387
+ better UV	210	3	37	6	74	378	1294	2383

$B_1 = 1024$ $B_2 = 7 \cdot 2^{14}$	ω	Baby-step dDBL	Baby-step dADD	Giant-step dDBL	Giant-step dADD	product **M**	product **M**	total **M**
CADO-NFS 2.3.0	210	3	37	12	505	2078	7859	13264
+ better UV	210	3	37	12	505	2078	7857	13262

New stage 2 in CADO-NFS

$B_1 = 256$ $B_2 = 2^{14}$	ω	Baby-step dDBL	Baby-step dADD	Giant-step dDBL	Giant-step dADD	product **M**	product **M**	total **M**
CADO-NFS 2.3.0	210	3	37	6	74	378	1298	2387
+ better UV	210	3	37	6	74	378	1294	2383
+ better giant step	210	3	37	36	39	378	1294	2323

$B_1 = 1024$ $B_2 = 7 \cdot 2^{14}$	ω	Baby-step dDBL	Baby-step dADD	Giant-step dDBL	Giant-step dADD	product **M**	product **M**	total **M**
CADO-NFS 2.3.0	210	3	37	12	505	2078	7859	13264
+ better UV	210	3	37	12	505	2078	7857	13262
+ better giant step	210	3	37	230	276	2078	7857	12978

New stage 2 in CADO-NFS

$B_1 = 256$ $B_2 = 2^{14}$	ω	Baby-step		Giant-step		product	total
		dDBL	dADD	dDBL	dADD	**M**	**M**	**M**
CADO-NFS 2.3.0	210	3	37	6	74	378	1298	2387
+ better UV	210	3	37	6	74	378	1294	2383
+ better giant step	210	3	37	36	39	378	1294	2323
+ using best ω	330	3	57	22	25	330	1280	2227

$B_1 = 1024$ $B_2 = 7 \cdot 2^{14}$	ω	Baby-step		Giant-step		product	total
		dDBL	dADD	dDBL	dADD	**M**	**M**	**M**
CADO-NFS 2.3.0	210	3	37	12	505	2078	7859	13264
+ better UV	210	3	37	12	505	2078	7857	13262
+ better giant step	210	3	37	230	276	2078	7857	12978
+ using best ω	798	3	135	49	74	890	7869	10273

Stage 2 on twisted Edwards curves

The theory is the same, except that in the formulæ, the X coordinates must be replaced by Y coordinates.
Description based on Miele's thesis and implementation by Ishii *et al.*.

They do not use any method to reduce the size of the set UV

Baby-step (assumes ω is even):

- compute $[\Delta u]Q$ for all Δu appearing as the difference of two consecutive values of $u \in U$;
- compute $[u]Q$ for $u \in U$ with one addition;
- compute $[\omega]Q$ using previously computed points.

Giant-step:

- compute $[v][\omega]Q$ for all $v \leq \max V$.

The product is computed using the second improvement but the homogenization is done in two steps costing $5(\#U + \#V) - 10$ multiplications instead of $4(\#U + \#V) - 6$ multiplications.

Stage 2 on twisted Edwards curves

$B_1 = 256$ $B_2 = 2^{14}$	ω	Baby-step + Giant-step				product		total
		DBL	DBL_ε	ADD	ADD_ε	**M**	**M**	**M**
Ishii *et al.*	420	1	22	19	50	187+238	1397	2538

$B_1 = 1024$ $B_2 = 7 \cdot 2^{14}$	ω	Baby-step + Giant-step				product		total
		DBL	DBL_ε	ADD	ADD_ε	**M**	**M**	**M**
Ishii *et al.*	1050	1	57	54	122	475+660	8458	11410

In the first example, using $\omega = 330$ reduces the total cost by 6**M**.
In the second example, $\omega = 1050$ is still the best ω after the two improvements.

Stage 2 on twisted Edwards curves

$B_1 = 256$ $B_2 = 2^{14}$	ω	Baby-step + Giant-step				product		total
		DBL	DBL_ε	ADD	ADD_ε	**M**	**M**	**M**
Ishii *et al.*	420	1	22	19	50	187+238	1397	2538
+ better UV	420	1	22	19	50	187+223	1305	2431

$B_1 = 1024$ $B_2 = 7 \cdot 2^{14}$	ω	Baby-step + Giant-step				product		total
		DBL	DBL_ε	ADD	ADD_ε	**M**	**M**	**M**
Ishii *et al.*	1050	1	57	54	122	475+660	8458	11410
+ better UV	1050	1	57	54	122	475+615	7840	10747

In the first example, using $\omega = 330$ reduces the total cost by 6**M**.
In the second example, $\omega = 1050$ is still the best ω after the two improvements.

Stage 2 on twisted Edwards curves

$B_1 = 256$ $B_2 = 2^{14}$	ω	Baby-step + Giant-step				product		total
		DBL	DBL_ε	ADD	ADD_ε	**M**	**M**	**M**
Ishii *et al.*	420	1	22	19	50	187+238	1397	2538
+ better UV	420	1	22	19	50	187+223	1305	2431
+ 1-step homogenization	420	1	22	19	50	330	1305	2351

$B_1 = 1024$ $B_2 = 7 \cdot 2^{14}$	ω	Baby-step + Giant-step				product		total
		DBL	DBL_ε	ADD	ADD_ε	**M**	**M**	**M**
Ishii *et al.*	1050	1	57	54	122	475+660	8458	11410
+ better UV	1050	1	57	54	122	475+615	7840	10747
+ 1-step homogenization	1050	1	57	54	122	874	7840	10531

In the first example, using $\omega = 330$ reduces the total cost by 6**M**.
In the second example, $\omega = 1050$ is still the best ω after the two improvements.