Faster cofactorization with ECM using mixed representations

Cyril Bouvier and Laurent Imbert

LIRMM, CNRS, Univ. Montpellier, France

IACR International Conference on Practice and Theory of Public-Key Cryptography

June 1–4 2020

Context – integer factorisation

Number Field Sieve (NFS): best known algorithm for factoring large integers and computing DL over finite fields

Current record (feb. 2020): RSA-250 (a 829-bit integer)

- Cofactorization: an important step in sieving phase of NFS ($\approx 1/3$ of the time for RSA-768)
- Goal: breaking into primes billions of medium-size integers
- Method of choice: Elliptic Curve Method (ECM) [H. Lenstra '85]

Background

Block generation (beyond NAF)

Block combination

Results and comparisons

ECM scalar multiplication

Step 1 of ECM: compute $[k]P$ where

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

Example with smoothness bound $B_1 = 32$

$$k = 2^5 \times 3^3 \times 5^2 \times 7 \times 11 \times \cdots \times 29 \times 31$$

Two naive options:

- evaluate $k \in \mathbb{Z}$ first
- accumulate $[\pi]P$ for each prime $\pi \leq B_1$ (with multiplicities)

ECM in the context of NFS cofactorisation

- medium-size integers (≈ 150 bits)
- B_1-values: small and fixed

  Ex. from CADO-NFS: $105 \leq B_1 \leq 8192$
- k is known in advance

Goal:

Design "optimal" algorithms for computing $[k]P$ for all B_1-values

Dixon and Lenstra's idea

Regroup some primes in "blocks" to reduce # ADD

$$k = \prod_{\pi \text{ prime} \leq B_1} \pi^{\lfloor \log_\pi(B_1) \rfloor} = \prod_i (\pi_{i_1} \times \cdots \times \pi_{i_s})$$

Example (using double-and-add, #ADD = HW - 1):

block	Hamming Weight
$\pi_1 = 1028107$	10
$\pi_2 = 1030639$	16
$\pi_3 = 1097101$	11
$\pi_1 \times \pi_2 \times \pi_3$	8

Dixon and Lenstra: blocks of at most 3 primes

Bos and Kleinjung's improvement

Generation of blocks of > 3 primes: too expensive for practical B_1-values

Opposite strategy:

generate a huge number of integers with very low Hamming weights and check for smoothness

Example of blocks for $B_1 = 32$:

- $1000000000100001_2 = 2^{16} + 2^5 + 1 = 7 \times 17 \times 19 \times 29$ ✓
- $1000000000010001_2 = 2^{16} + 2^4 + 1 = 3 \times 21851$ ✗
- $100000000001\bar{1}_2 = 2^{12} - 1 = 3^2 \times 5 \times 7 \times 13$ ✓

Which curve model is best suited to ECM?

No clear answer!

	Montgomery	(twisted) Edwards
Coord. system	XZ-only	projective
DBL	$++$	$+$
TPL	$+$	$+$
ADD	differential	$+$
Scalar mult.	Lucas chains	D&A, wNAF, etc.

Theorem [Berstein et al.]: Every twisted Edwards curve is birationally equivalent to a Montgomery curve

Our contribution

A good mix of Montgomery and Edwards curves

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

  New Op ADD_M:

  P_1, P_2 in Edwards $\rightarrow$ $P_1 + P_2$ in Montg. XZ (cost: 4M)
- finish the computation on the Montgomery curve (including step 2 of ECM)

Extension and improvement of Bos and Kleinjung's algorithm

- with blocks of various types (beyond NAF)
- a better (nearly optimal) block combination algorithm

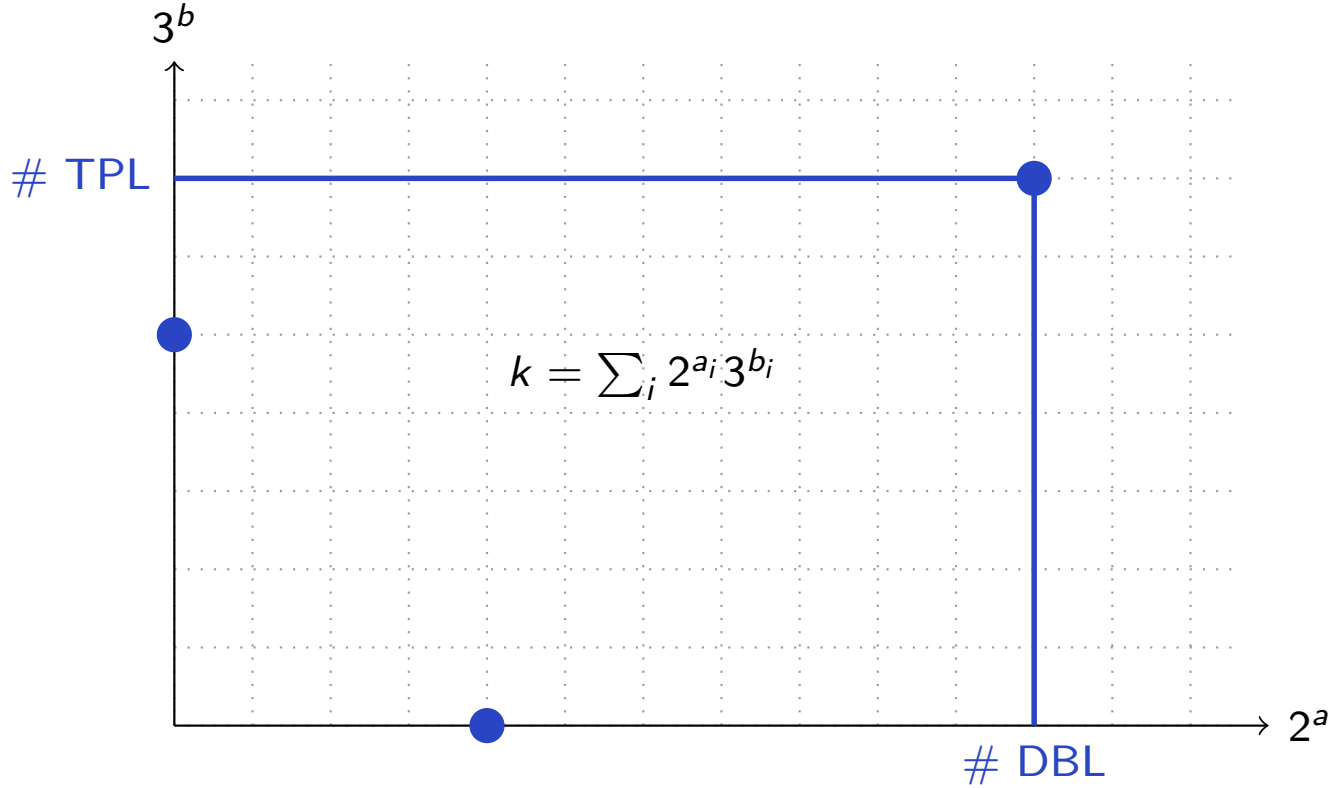
Background

Block generation (beyond NAF)

Block combination

Results and comparisons

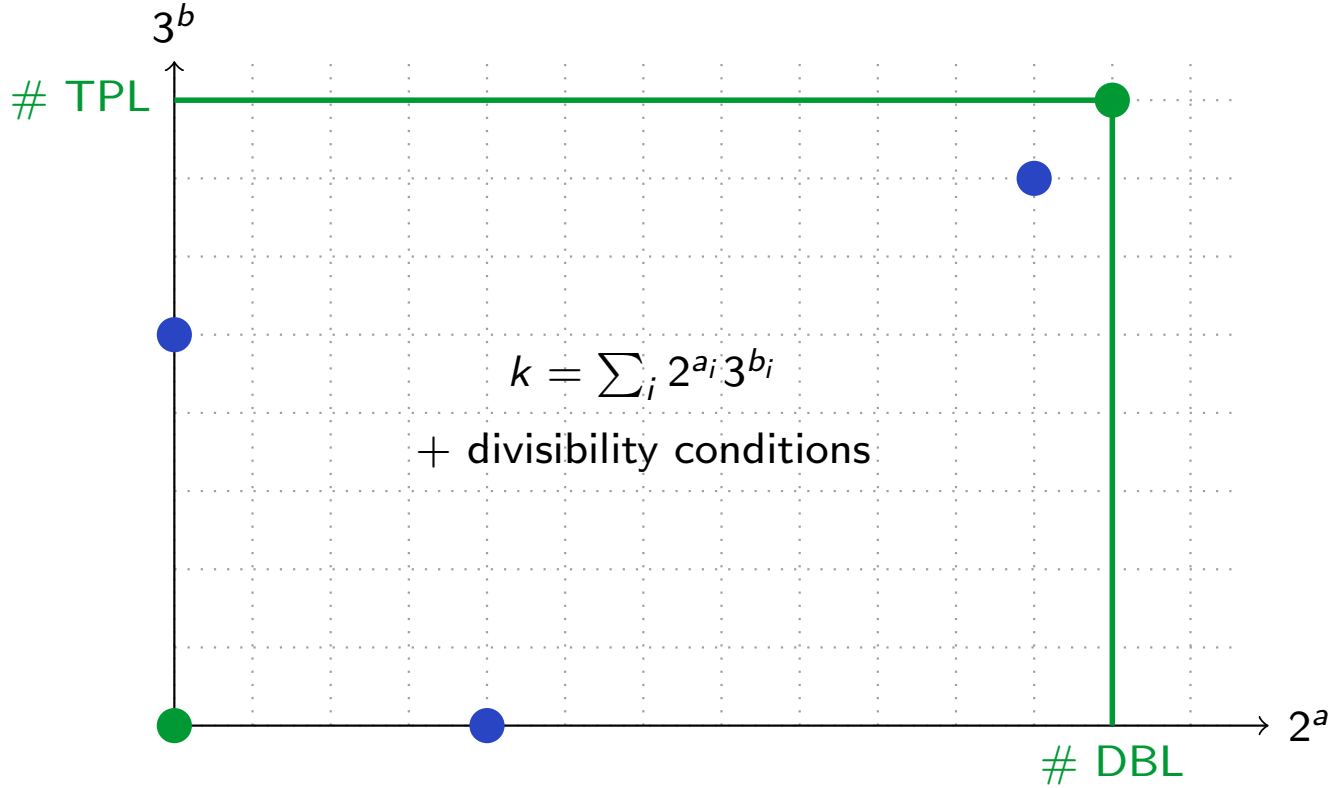
Edwards curves – Double-base expansions/chains



double-base expansion: $2^{11}3^7 + 2^4 - 3^5 = 103 \times 67 \times 59 \times 11$

11 DBL, 7 TPL, 2 ADD, precomp: 3 points

Edwards curves – Double-base expansions/chains



double-base expansion: ● $2^{11}3^7 + 2^4 - 3^5 = 103 \times 67 \times 59 \times 11$

11 DBL, 7 TPL, 2 ADD, precomp: 3 points

double-base chain: ● $2^{12}3^8 - 1 = 73 \times 71 \times 61 \times 17 \times 5$

12 DBL, 8 TPL, 1 ADD, no storage

Montgomery curves – Lucas chains

Differential addition (DADD) : $P,\ Q,\ (P-Q) \longrightarrow P+Q$

Lucas chains: $(1 = c_0, c_1, \ldots, c_t = k)$ s.t. $\ell > 0 \Rightarrow c_\ell = c_i + c_j$
and either $c_i = c_j$ (DBL) or $|c_i - c_j| = c_m$ for some $i, j, m < k$ (DADD)

Lucas chains can be computed using Montgomery's PRAC algorithm

rule A: sequence of curve ops.
rule B: sequence of curve ops.
rule C: sequence of curve ops.
⋮
rule J: sequence of curve ops.

Inverting PRAC:

Generate short words on the alphabet {A,B,C,...,J}, i.e. short Lucas chains
Compute corresponding integer and test for smoothness

Block generation

Similar to Bos and Kleinjung's approch

A very large number of blocks of each type for $B_1 = 2^{13}$

Filtered out after smoothness test and redundant elimination

Block type	gross	net	time (hours)
double-base expansions for various # DBL, #TPL, #ADD	10^{12}	10^7	1000
double-base chains for various # DBL, #TPL, #ADD	10^{13}	10^9	9000
Lucas chains	10^{19}	5.10^6	700

No unnecessary computation: no block was generated more than once

Background

Block generation (beyond NAF)

Block combination

Results and comparisons

Goal

Remember that the goal is to efficiently compute the scalar multiplication by $k = \prod_{\pi \text{ prime} \leq B_1} \pi^{\lfloor \log(B_1)/\log(\pi) \rfloor}$ in ECM.

Example (ECM for $B_1 = 32$)

The scalar multiplication of ECM for $B_1 = 32$ can be done using 8 blocks:

- 1 double-base chain on the twisted Edwards curve
  [to compute the scalar product by $3 \times 11 \times 31$]
- 7 Lucas chains on the corresponding Montgomery curve
  [to compute the scalar product by 2^3, $3^2 \times 5 \times 7$, 5×13, 29, 23, 19, 17]

Combination algorithm

Find the subset of all the computed blocks with the smallest "cost" such that the product of the integers represented by these blocks is exactly k.

The "cost" is the sum of the arithmetic cost of all the blocks.

Bos and Kleinjung's combination algorithm

Bos and Kleinjung used a greedy algorithm to combine blocks.

Very fast. Generates good but non-optimal solution.

Uses two values to choose the "best" blocks to add in the solution set:

- the ratio number of doublings / number of additions (the larger the better)
- a score function designed to favor blocks with a large numbers of large factors

They also proposed a randomized version of their algorithm.

Adapting Bos and Kleinjung's algorithm to our setting

Ratio number of doublings / number of additions does not readily apply to our setting because

- we also use triplings
- we use both twisted Edwards and Montgomery curves where additions and doublings have different costs

We also observed that the score does not always achieve its goal to favor blocks with large factors.

For example, it favors blocks with 3 large factors compared to a block with 3 large factors and 3 medium ones.

Our algorithm

Found no suitable replacement for the score function

We try to sort the blocks by arithmetic cost per bit but it does not yield better results

A complete exhaustive search is totally out of reach, even for small B_1-values.

An almost exhaustive solution:

- Shrink the enumeration depth with an upper bound on the number of blocks in a solution set $\mathcal{S}$.
  We loose solutions, but we hope that the best solution has a small number of blocks.
- Reduce the enumeration width at each step using the knowledge of an upper bound on the minimal cost.
  Here, we do not loose any solution.

Exploiting an upper bound on the minimal cost

An upper bound on the minimal cost can be found with any method (Bos and Kleinjung's algorithm, *double-and-add*, ...)

Using this knowledge, we are able to compute a upper bound on the arithmetic cost per bit of a block that can be added to the current solution set.

Our algorithm:

- Sort the set of all generated blocks by increasing value of the arithmetic cost per bit
- Enumerate, depth-first, all subsets of blocks of size less than a given bound
  - at each step of the enumeration, compute the bound on the arithmetic cost per bit and discard inadmissible blocks
  - the bound on the arithmetic cost of the best solution set can be updated during the algorithm

Background

Block generation (beyond NAF)

Block combination

Results and comparisons

Example: best solution found for $B_1 = 105$

Blocks	Type	
$73 \times 71 \times 61 \times 17 \times 5$	Double-base chains	$2^{12} \cdot 3^8 - 1$
$97 \times 43 \times 37 \times 31 \times 13 \times 7 \times 5$		$2^{12} \cdot 3^{12} - 1$
$89 \times 53 \times 29 \times 23$		$2^{20} \cdot 3 + 2^9 - 1$
$101 \times 83 \times 79 \times 19$	Double-base expansions	$2^{22} \cdot 3 - 2^5 + 3$
$103 \times 67 \times 59 \times 11$		$2^{11} \cdot 3^7 + 2^4 - 3^5$
Switch to Montgomery curve		
3^2	Lucas chains	
3^2		
7		
47		
41		
2^6		
Total arithmetic cost		1144 multiplications and squares

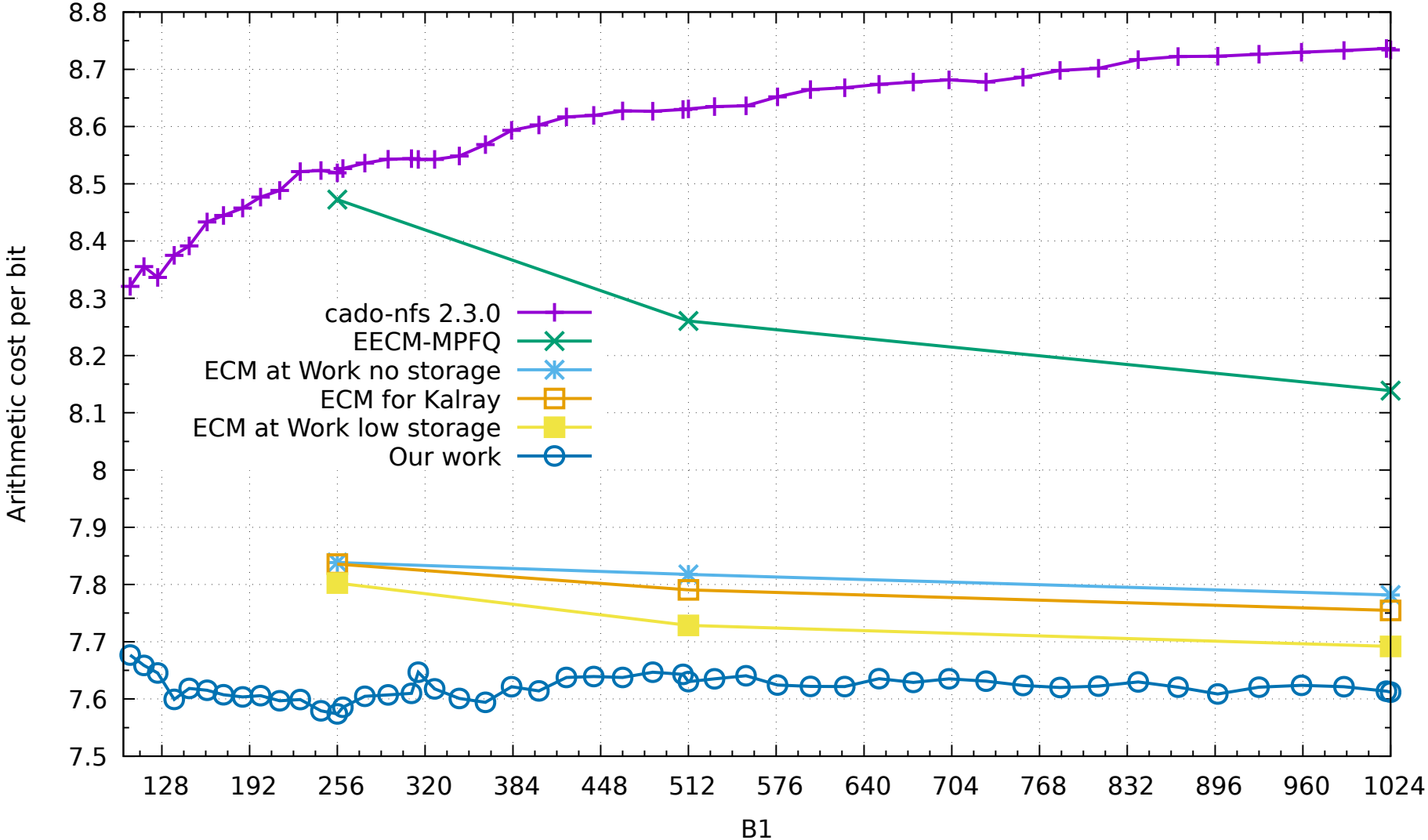
Cost comparison – number of multiplications

	$B_1 =$	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

Number of modular multiplication (**M**) for various implementations of ECM 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 – arithmetic cost per bit



Implementation

We implemented in CADO-NFS our new algorithm for the scalar multiplication.

Comparison on large computations with CADO-NFS:

- we run parts of the cofactorization step of NFS for RSA-200 and RSA-220
- time decreased by 5% to 10%
- corresponds to our theoretical estimates

Conclusions

We improved the implementation of ECM in the context of NFS cofactorization

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

Our ECM implementation uses:

- both twisted Edwards curves and Montgomery curves
- a new addition-and-switch operation
- uses double-base expansions and chains and PRAC-generated Lucas chains

Results and source code are available at

`http://eco.lirmm.net/double-base_ECM/`