Comparing the Difficulty of Factorization and Discrete Logarithm: a 240-digit Experiment

Solving RSA-240, DLP-240, RSA-250

Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic,
Nadia Heninger, Emmanuel Thomé, Paul Zimmermann

FB: Ministère de l'Éducation Nationale, Université de Limoges
PG+AG+ET+PZ: Université de Lorraine, CNRS, Inria, Nancy
NH: University of California, San Diego

August 21, 2020 – CRYPTO 2020

Plan

How does one choose key sizes?

When setting up crypto, a major decision is the key size.

- Efficiency: want shorter keys.
- Security: want longer keys.

A compromise is needed. An end user might...

- trust the manufacturer to do The Right Thing.
- check that it abides by recommendations by regulatory bodies (NIST, ANSSI, BSI, ...).

Tricky questions for public-key crypto

How does one assess the hardness of cryptanalysis, for key sizes that are (fortunately) out of its reach?
How does one make this assessment convincing?

We need hard facts

Predictions can only be based on state-of-the-art software implementation performance.

We need actual software that is fit for large sizes, together with convincing computational results.

- Explore algorithmic ideas that pay off only for large sizes.
- Explore scalability, try to address stumbling blocks.
- Harness large computing power, show that this is more than just theory.
- Make our work reproducible.

IF versus FF-DLP

We look at two important problems:

- IF: Integer Factorization;
- FF-DLP: Finite Field Discrete Logarithm Problem.

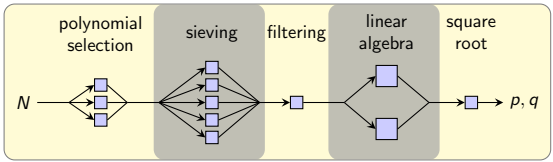
The appreciation of their relative difficulty is hard to do because IF and FF-DLP records are usually done out of sync.

Common belief: for similar key sizes, FF-DLP is a lot harder than IF.

Plan

Summary of NFS

The NFS algorithm (1990) proceeds through many steps.
Example workflow in the IF case.



Computational requirements are diverse.

- Sieving (relation collection) is the most expensive.
  It can be massively distributed.
- (Sparse) Linear algebra comes second.
  It is somewhat cheaper, but needs expensive hardware.

What kind of *relations* does NFS collect?

Polynomial selection finds f with a known root $m \mod N$.
Let $\mathbb{Q}(\alpha)$ be the number field defined by f.

Bird's eye view of strategy for factoring

- Search for pairs of integers (a, b) such that

$$\underbrace{a - bm}_{\text{(an integer)}} \quad \text{and} \quad \underbrace{a - b\alpha}_{\text{(an ideal in } \mathbb{Q}(\alpha))}$$

  are both smooth: they factor into small things. Pairs yield relations.
- Combine relations so that all multiplicities are even.
  We (almost) have squares on both sides.
- With further (easy) work, we find many equalities of squares:
  $u^2 \equiv v^2 \mod N$ leads to factors of N with probability at least $\frac{1}{2}$.

NFS also works for FF-DLP

NFS works similarly for the discrete logarithm.

- N becomes p. Note that $\mathbb{Z}/p\mathbb{Z}$ is a field.
- Both sides are number fields. Not an issue.
- FF-DLP version of NFS is no longer a story of finding squares.
- We no longer seek even valuations and linear algebra over $\mathbb{Z}/2\mathbb{Z}$, but linear algebra over $\mathbb{Z}/\ell\mathbb{Z}$, with ℓ a (large) prime factor of $p-1$.
  It's harder.
- But the general pattern remains unchanged.

Plan

Collecting relations

Relation collection is the most expensive step of NFS.

Description of relation collection

1. How do we divide the work?
2. How do we find smooth $a - bm$ and $a - b\alpha$?
3. How do we choose parameters so that the cost of linear algebra remains under control?

1. (many) needles in a (huge) haystack

Searching a space of size (say) 2^{65} takes long.

- Trivial strategy (loop over a, loop over b) has unstable yield and does not work well.
- Better: constrain a factor q in one of the factorizations.
  - ✓ Independent tasks per q.
  - ✓ Yield is stable.
  - ✓ The prescribed factor is one thing less to find!

  (old folklore; records have been doing this for decades.)

  ($\Rightarrow$ special-q sieving, lattice sieving, sieving by vectors.)

2. Finding smooth (a, b)

We have fixed a q.

We explore many (a, b) such that q appears somewhere.

We want $a - bm$ and $a - b\alpha$ to be smooth.

Strategy depends on potential prime factors p.

A prime should appear either often, or very rarely.

- below some bound, for $p < B$, strive to find all pairs (a, b) such that p appears in the factorization.
  We typically use a process called sieving.
- "large primes" (LPs) such that $B \leq p < L$:
  allowed if we happen to find them.
  Limit to a few LPs per relation (e.g., 2, sometimes 3).

The relations that we like to see

✓	$5^2 \cdot 11 \cdot 23 \cdot 287093 \cdot 870953 \cdot 20179693 \cdot 28306698811 \cdot 47988583469$	$2^3 \cdot 5 \cdot 7 \cdot 13 \cdot 31 \cdot 61 \cdot 14407 \cdot 26563253 \cdot 86800081 \cdot 269845309 \cdot 802234039 \cdot 1041872869 \cdot 5552238917 \cdot 12144939971 \cdot 15856830239$
✓	$3 \cdot 1609 \cdot 77699 \cdot 235586599 \cdot 347727169 \cdot 369575231 \cdot 9087872491$	$2^3 \cdot 3 \cdot 5 \cdot 13 \cdot 19 \cdot 23 \cdot 31 \cdot 59 \cdot 239 \cdot 3989 \cdot 7951 \cdot 2829403 \cdot 31455623 \cdot 225623753 \cdot 811073867 \cdot 1304127157 \cdot 78955382651 \cdot 129320018741$
✓	$5 \cdot 1381 \cdot 877027 \cdot 15060047 \cdot 19042511 \cdot 11542780393 \cdot 13192388543$	$2^4 \cdot 5 \cdot 13 \cdot 31 \cdot 59 \cdot 823 \cdot 2801 \cdot 26539 \cdot 2944817 \cdot 3066253 \cdot 87271397 \cdot 108272617 \cdot 386616343 \cdot 815320151 \cdot 1361785079 \cdot 12322934353$
✓	$2^3 \cdot 5^2 \cdot 173 \cdot 971 \cdot 613909489 \cdot 929507779 \cdot 1319454803 \cdot 2101983503$	$2^7 \cdot 3^2 \cdot 5 \cdot 29 \cdot 1021 \cdot 42589 \cdot 190507 \cdot 473287 \cdot 31555663 \cdot 654820381 \cdot 802234039 \cdot 19147596953 \cdot 23912934131 \cdot 52023180217$
✗	$2^2 \cdot 15193 \cdot 232891 \cdot 19514983 \cdot 139295419 \cdot 540260173 \cdot 606335449$	$2^2 \cdot 3^4 \cdot 13 \cdot 19 \cdot 74897 \cdot 1377667 \cdot 55828453 \cdot 282012013 \cdot 802234039 \cdot 3350122463 \cdot 35787642311 \cdot 37023373909 \cdot 128377293101$
✗	$2^2 \cdot 5^4 \cdot 439 \cdot 1483 \cdot 13121 \cdot 21383 \cdot 67751 \cdot 452059523 \cdot 33099515051$	$2^2 \cdot 3^3 \cdot 11 \cdot 13 \cdot 19 \cdot 5023 \cdot 3683209 \cdot 98660459 \cdot 802234039 \cdot 1506372871 \cdot 4564625921 \cdot 27735876911 \cdot 32612130959 \cdot 45729461779$

small primes: abundant $\rightarrow$ dense column in the matrix

large primes: rare $\rightarrow$ sparse colum, limit to 2 or 3 on each side.

The relations that we like to see

✓ $5^2 \cdot 11 \cdot 23 \cdot 287093 \cdot 870953 \cdot 20179693 \cdot 28306698811 \cdot 47988583469$

✓ $3 \cdot 1609 \cdot 77699 \cdot 235586599 \cdot 347727169 \cdot 369575231 \cdot 9087872491$

✓ $5 \cdot 1381 \cdot 877027 \cdot 15060047 \cdot 19042511 \cdot 11542780393 \cdot 13192388543$

✓ $2^3 \cdot 5^2 \cdot 173 \cdot 971 \cdot 613909489 \cdot 929507779 \cdot 1319454803 \cdot 2101983503$

✗ $2^2 \cdot 15193 \cdot 232891 \cdot 19514983 \cdot 139295419 \cdot 540260173 \cdot 606335449$

✗ $2^2 \cdot 5^4 \cdot 439 \cdot 1483 \cdot 13121 \cdot 21383 \cdot 67751 \cdot 452059523 \cdot 33099515051$

$2^3 \cdot 5 \cdot 7 \cdot 13 \cdot 31 \cdot 61 \cdot 14407 \cdot 26563253 \cdot 86800081 \cdot 269845309 \cdot 802234039 \cdot 1041872869 \cdot 5552238917 \cdot 12144939971 \cdot 15856830239$

$2^3 \cdot 3 \cdot 5 \cdot 13 \cdot 19 \cdot 23 \cdot 31 \cdot 59 \cdot 239 \cdot 3989 \cdot 7951 \cdot 2829403 \cdot 31455623 \cdot 225623753 \cdot 811073867 \cdot 1304127157 \cdot 78955382651 \cdot 129320018741$

$2^4 \cdot 5 \cdot 13 \cdot 31 \cdot 59 \cdot 823 \cdot 2801 \cdot 26539 \cdot 2944817 \cdot 3066253 \cdot 87271397 \cdot 108272617 \cdot 386616343 \cdot 815320151 \cdot 1361785079 \cdot 12322934353$

$2^7 \cdot 3^2 \cdot 5 \cdot 29 \cdot 1021 \cdot 42589 \cdot 190507 \cdot 473287 \cdot 31555663 \cdot 654820381 \cdot 802234039 \cdot 19147596953 \cdot 23912934131 \cdot 52023180217$

$2^2 \cdot 3^4 \cdot 13 \cdot 19 \cdot 74897 \cdot 1377667 \cdot 55828453 \cdot 282012013 \cdot 802234039 \cdot 3350122463 \cdot 35787642311 \cdot 37023373909 \cdot 128377293101$

$2^2 \cdot 3^3 \cdot 11 \cdot 13 \cdot 19 \cdot 5023 \cdot 3683209 \cdot 98660459 \cdot 802234039 \cdot 1506372871 \cdot 4564625921 \cdot 27735876911 \cdot 32612130959 \cdot 45729461779$

small primes: abundant $\rightarrow$ dense column in the matrix
large primes: rare $\rightarrow$ sparse colum, limit to 2 or 3 on each side.

Before linear algebra, the filtering step tries to do as many cheap combinations as it can, so as to get a smaller matrix.

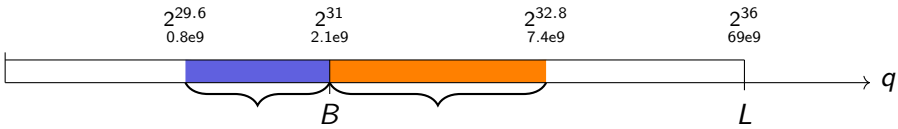
3. Paying attention to the combination cost

Relations with 2 LPs or less are a blessing.

- They easily participate in cheap combinations.
- If we have only 2-LP relations, filtering will get rid of most of them. We are left with a number of primes to combine that is roughly the number of primes below B.
- Caveat: two sides to deal with.

We must pay attention to q as well! How does it compare to B?

Strategy for RSA-240



$q < B$: allow
2 LPs on side 0,
3 LPs on side 1.

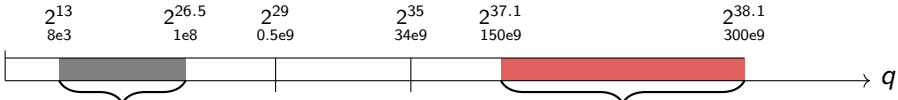
$B \leq q < L$: allow 2 LPs on each side.
(q counts as an extra LP on side 1.)

This strategy makes it easy to get rid of most $p \geq B$ on side 0 before we enter linear algebra proper.

We still have many on side 1, but that is not too bad because linear algebra in the factoring context is reasonable.

Strategy for DLP-240

For DLP-240, we used composite q, to avoid the disadvantage of having q in the LP range.



This strategy was efficient in reducing the combination work to essentially primes $p < B$ only.

Alternative to sieving

Another important ingredient that we used. Fun way to find the B-smooth part of many integers (say, many $a - bm$'s) (Bernstein, 2000):

- Multiply all primes together! $P = \prod_{p<B} p$.
- Multiply all integers together! $M = \prod_{(a,b)\in\mathcal{S}}(a - bm)$. (keeping track of the tree of subproducts)

Alternative to sieving

Another important ingredient that we used. Fun way to find the B-smooth part of many integers (say, many $a - bm$'s) (Bernstein, 2000):

- Multiply all primes together! $P = \prod_{p<B} p$.
- Multiply all integers together! $M = \prod_{(a,b)\in\mathcal{S}}(a - bm)$. (keeping track of the tree of subproducts)
- Compute P mod M, then P mod the two halves and so on, down to $\{P \bmod (a - bm)\}_{(a,b)\in\mathcal{S}}$.

This has pros and cons.

- ✓ Asymptotically fast with FFT-like algorithms.
- ✓ Finds all primes $< B$ (so does sieving).
- ✗ Requires some memory, and not trivial to parallelize.
- ✓ But allows to save memory in other steps.

Plan

Rules of the game

Everything reduces to linear algebra

- Integer Factorization: combining relations so as to obtain even valuations is a linear algebra problem, over $\mathbb{Z}/2\mathbb{Z}$.
- FF-DLP: not the same goal, but still a linear algebra problem, this time over $\mathbb{Z}/\ell\mathbb{Z}$ where ℓ has several hundred bits.

Matrix dimensions
- RSA-240: nrows=282M; 200 non-zero/row.
- DLP-240: nrows=37M; 250 non-zero/row.

As the matrix is sparse, we use an iterative algorithm.
Key operation is SpMV: sparse matrix times vector.

Better scaling

We use the block Wiedemann algorithm (1994).

- iterative, with n (shorter) independent sequences.
  (Almost) linear scaling in the iterative part, with no communication.
- work can be reconciled by computing a generator ($\approx \chi$).
  The more sequences we use, the more expensive it gets.

Cost metrics with n sequences (assuming $n < \log \text{nrows}$)

	Time	Memory	main operations
Per sequence	$O(\text{nrows}^2/n)$ n-way distributed	$O(\text{nrows})$	`movq`, `addq`
Generator	$\widetilde{O}(n \cdot \text{nrows})$	$O(n \cdot \text{nrows})$	$*$ in $\mathcal{M}_{n,n}(\mathbb{Z}/\ell\mathbb{Z}[x])$

Plan

The CADO-NFS software

We used the CADO-NFS software.

- Important software development effort since 2007.
- 250k lines of C/C++ code.
- Open source (LGPL), open development model (gitlab).

Regarding relation collection only:

- An important part of CADO-NFS (60k lines)
- Significant improvements since 2016.
  - improved parallelism: strive to get rid of scheduling bubbles;
  - versatility: large freedom in parameter selection;
  - prediction of behaviour and yield: essential for tuning.

Work on linear algebra

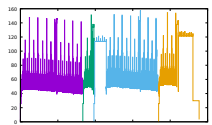
Linear algebra is a big part of CADO-NFS (60k lines C/C++).

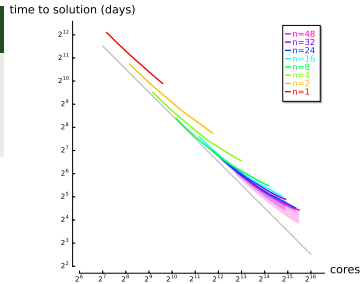
- SpMV uses MPI+threads, some low-level assembly.
  2016: big performance improvements.
- Generator step uses asymptotically fast FFT-based algorithms, MPI, threads, ...
  2019–: better parallelization, more flexible (still WIP).

We can achieve fairly good scaling

- Use several sequences.
- Do SpMV in parallel as well.

RAM in generator step:
be careful!





Computer resources used

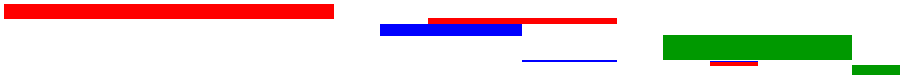
We used several computer clusters. Mostly recent hardware.

- Grid5000 clusters (FR) in best-effort mode.
- Hardware from University of Pennsylvania.
  (later moved to UCSD) (the move took time).
- Campus computer cluster in Nancy, France.
- 32M core-hours compute allocation on EU PRACE infrastructure.

Note: each platform comes with its usage specificities (hardware, software, policies, faults, ...).

Plan

Approximative timeline and core-hours



2018/08 - 2019/03	**DLP-240** relation collection.	21M c·h
	4k cores working in parallel.	
2019/05 - 2019/08	**DLP-240** linear algebra (sequences)	5M c·h
2019/04 - 2019/06	**RSA-240** relation collection.	7M c·h
	4.3k cores working in parallel.	
2019/10 - 2020/02	**RSA-250** relation collection.	21M c·h
	12k cores working in parallel.	
2019/07 - 2019/08	**RSA-240** linear algebra (sequences)	0.6M c·h
2019/11	**RSA-240** linear algebra (wrap up)	0.1M c·h
	DLP-240 linear algebra (wrap up)	0.7M c·h
2020/02	**RSA-250** linear algebra	2M c·h

caveat: time windows often include partially idle periods

Total cost

Aggregated cost in physical core-years (c·y) and core-hours (c·h) (platform details in paper).

	sieving	matrix
RSA-240	800 c·y (7Mc·h)	83 c·y (0.7Mc·h)
DLP-240	2,400 c·y (21Mc·h)	625 c·y (6Mc·h)
RSA-250	2,450 c·y (21Mc·h)	250 c·y (2Mc·h)

Extensive information on the computational data, the parameters, and how to reproduce (parts of) our computation can be found at
https://gitlab.inria.fr/cado-nfs/records/.

Conclusions

- More than just records, we developed efficient parameterization strategies for further computations.
- We developed an extensive simulation framework to guide the parameter choices. Not perfect.
- We show that our implementation scales well and can tackle larger problems. No technology barrier at this point.

Comparisons:

- Comparison with previous record (DLP-768, 232 digits, 2016): On identical hardware, our DLP-240 computation would have taken less time than the 232-digits computation.
- FF-DLP is not much harder than integer factoring.

For future projects, we intend to keep the focus on our capacity to anticipate the computational cost, and to harness large computing power.