HPC in Cryptanalysis

A short tutorial

Antoine Joux

Eurocrypt 2012
April 16th,2012

Why “HPC in Cryptanalysis” ?

Why "HPC in Cryptanalysis" ?

- Historical link

Why “HPC in Cryptanalysis” ?

- Historical link

THE NATIONAL MUSEUM OF COMPUTING

BLETCHLEY PARK SCIENCE & INNOVATION CENTRE

- Background activity in support of research

Why "HPC in Cryptanalysis" ?

- Historical link

THE NATIONAL MUSEUM OF COMPUTING

BLETCHLEY PARK SCIENCE & INNOVATION CENTRE

- Background activity in support of research
- Fun (but sometime frustrating)

How special are computations in Cryptanalysis ?

How special are computations in Cryptanalysis ?

- Aimed at record breaking / new algorithms benchmarking

How special are computations in Cryptanalysis ?

- Aimed at record breaking / new algorithms benchmarking
- No real need for reusability

How special are computations in Cryptanalysis ?

- Aimed at record breaking / new algorithms benchmarking
- No real need for reusability
- Have to be performed on whatever is available

How special are computations in Cryptanalysis ?

- Aimed at record breaking / new algorithms benchmarking
- No real need for reusability
- Have to be performed on whatever is available
- Computations are easy to check

Main steps

Main steps

- Algorithmic starting point

Main steps

- Algorithmic starting point
  - Validation by toy implementation

Main steps

- Algorithmic starting point
  - Validation by toy implementation

- Find computing power / Choose target computation

Main steps

- Algorithmic starting point
  - Validation by toy implementation
- Find computing power / Choose target computation
- Program / Debug / Optimize

Main steps

- Algorithmic starting point
  - Validation by toy implementation
- Find computing power / Choose target computation
- Program / Debug / Optimize
- Run and Manage computation

Starting points : personal sample

Starting points : personal sample

- Lattice reduction and applications

Starting points : personal sample

- Lattice reduction and applications
- Collisions and multicollisions

Starting points : personal sample

- Lattice reduction and applications
- Collisions and multicollisions
- Elliptic curves, pairings, volcanoes

Starting points : personal sample

- Lattice reduction and applications
- Collisions and multicollisions
- Elliptic curves, pairings, volcanoes
- Index Calculus

Starting points : personal sample

- Lattice reduction and applications
- Collisions and multicollisions
- Elliptic curves, pairings, volcanoes
- Index Calculus
- Decomposition algorithms (Knapsacks, codes)

Starting points : personal sample

- Lattice reduction and applications
- Collisions and multicollisions
- Elliptic curves, pairings, volcanoes
- Index Calculus
- Decomposition algorithms (Knapsacks, codes)
- Gröbner bases

Stopping at toy implementations

Stopping at toy implementations

- Pairings
  - *Comparing the MOV and FR Reductions in E. C. Crypto*
    Harasama, Shikata, Suzuki, Imai
    $\Rightarrow$ Faster implementation using Miller's technique
  - Can be used constructively: Tripartite Diffie-Hellman

Stopping at toy implementations

- Pairings
  - *Comparing the MOV and FR Reductions in E. C. Crypto* Harasama, Shikata, Suzuki, Imai
    $\Rightarrow$ Faster implementation using Miller's technique
  - Can be used constructively: Tripartite Diffie-Hellman

- Volcanoes
  - *Pairing the volcano,* Ionica, J.

Finding computing power

Finding computing power

- Old-fashioned technique: Use/buy dedicated local machines
  - Easy to arrange (assuming funding available)
  - Good control of the architecture choice
  - Control on the availability of the computing resources
  - Not easy to scale

Finding computing power

- Old-fashioned technique: Use/buy dedicated local machines
  - Easy to arrange (assuming funding available)
  - Good control of the architecture choice
  - Control on the availability of the computing resources
  - Not easy to scale

- Email computations: Use idle cycles on desktop
  - Total available power is potentially huge
  - No control on choice of architecture or availability
  - Very limited communication bandwidth
  - Need to deal with “adversary” ressources
  - Need for a very user-friendly client

Finding computing power

Finding computing power

- Apply for power on HPC ressources
  - Very high-end dedicated computers
  - Fast communication
  - Need to use the existing architecture
  - Job management in a multi-user context is hard
  - Challenge: adapt to the massively parallel environment

Finding computing power

- Apply for power on HPC ressources
  - Very high-end dedicated computers
  - Fast communication
  - Need to use the existing architecture
  - Job management in a multi-user context is hard
  - Challenge: adapt to the massively parallel environment
- HPC in the Cloud

Choosing a target

Choosing a target

- Quality of target:
  - Proof of concept only
  - Real size demo
  - Attack cryptographic size parameters or record

Choosing a target

- Quality of target:
  - Proof of concept only
  - Real size demo
  - Attack cryptographic size parameters or record

- Reasonable feasability assurance

Proof of concept case

Proof of concept case

- *Differential collisions in SHA-0*, Chabaud, J.
  Full collision out of reach: Demo collisions
  - 80-rounds on partially linearized functions
  - 35-rounds on SHA-0

Proof of concept case

- *Differential collisions in SHA-0*, Chabaud, J.
  Full collision out of reach: Demo collisions
  - 80-rounds on partially linearized functions
  - 35-rounds on SHA-0

- *New generic algorithms for hard knapsacks.*
  Howgrave-Graham, J.
  Improved generic algorithms for hard knapsacks.
  Becker, Coron, J.

Proof of concept case

- *Differential collisions in SHA-0*, Chabaud, J.
  Full collision out of reach: Demo collisions
    - 80-rounds on partially linearized functions
    - 35-rounds on SHA-0

- *New generic algorithms for hard knapsacks.*
  Howgrave-Graham, J.
  Improved generic algorithms for hard knapsacks.
  Becker, Coron, J.

- *Decoding random binary linear codes in $2^{n/20}$.*
  Becker, J., May, Meurer

Medium case

Medium case

- *A practical attack against knapsack based hash functions*
  Granboulan, J. (1994)
  14 h single CPU, 25% success rate

Medium case

- *A practical attack against knapsack based hash functions* Granboulan, J. (1994)
  14 h single CPU, 25% success rate
- *Cryptanalysis of PKP: A new approach* Jaulmes, J. (2001)
  Full run 125 CPU.years (partially done)
  Reduced memory

Medium case

- *A practical attack against knapsack based hash functions*
  Granboulan, J. (1994)
  14 h single CPU, 25% success rate
- *Cryptanalysis of PKP: A new approach* Jaulmes, J. (2001)
  Full run 125 CPU.years (partially done)
  Reduced memory
- *Fast correlation attacks: an algorithmic point of view*,
  Chose, J., Mitton (2002)
  Reduced memory, demo on 40 bits LFSR, a few CPU days

Medium case

- *A practical attack against knapsack based hash functions*
  Granboulan, J. (1994)
  14 h single CPU, 25% success rate
- *Cryptanalysis of PKP: A new approach* Jaulmes, J. (2001)
  Full run 125 CPU.years (partially done)
  Reduced memory
- *Fast correlation attacks: an algorithmic point of view*,
  Chose, J., Mitton (2002)
  Reduced memory, demo on 40 bits LFSR, a few CPU days
- *Elliptic curve discrete logarithm problem over small degree extension fields* J., Vitse (JoC 2011)
  Adapted version of GB computations

The coding phase for records (personal view)

The coding phase for records (personal view)

- Keep it simple, stupid
  - Avoid fancy languages, remain at low-level

The coding phase for records (personal view)

- Keep it simple, stupid
  - Avoid fancy languages, remain at low-level
  - Avoid Libraries

The coding phase for records (personal view)

- Keep it simple, stupid
  - Avoid fancy languages, remain at low-level
  - Avoid Libraries
  - Avoid creeping featurism

The coding phase for records (personal view)

- Keep it simple, stupid
  - Avoid fancy languages, remain at low-level
  - Avoid Libraries
  - Avoid creeping featurism
  - Don't care too much about portability/reusability

The coding phase for records (personal view)

- Keep it simple, stupid
  - Avoid fancy languages, remain at low-level
  - Avoid Libraries
  - Avoid creeping featurism
  - Don't care too much about portability/reusability
  - Changes/Adaptations should be simple

The coding phase for records (personal view)

- Keep it simple, stupid
  - Avoid fancy languages, remain at low-level
  - Avoid Libraries
  - Avoid creeping featurism
  - Don't care too much about portability/reusability
  - Changes/Adaptations should be simple
- Optimization
  - Don't optimize non-critical parts
  - Don't over-optimize

The coding phase for records (personal view)

- Keep it simple, stupid
  - Avoid fancy languages, remain at low-level
  - Avoid Libraries
  - Avoid creeping featurism
  - Don't care too much about portability/reusability
  - Changes/Adaptations should be simple
- Optimization
  - Don't optimize non-critical parts
  - Don't over-optimize
- Main rule: avoid nasty surprises
  - Program from scratch
  - Conservative and defensive programming

Running the computation

Running the computation

- Tedious and difficult step

Running the computation

- Tedious and difficult step
- Scale up slowly to the intended size

Running the computation

- Tedious and difficult step
- Scale up slowly to the intended size
- Expect problems, software can fail
  - Easy phases don't scale well: Need to reprogram them on the fly
  - Rare bugs can be hard to detect: Check intermediate data

Running the computation

- Tedious and difficult step
- Scale up slowly to the intended size
- Expect problems, software can fail
  - Easy phases don't scale well: Need to reprogram them on the fly
  - Rare bugs can be hard to detect: Check intermediate data
- Expect problems, hardware can fail
  - Power down risk: Need ability to restart computation
  - Availability problems: Avoid tight schedule
  - Hardware faults can damage computations
    Check intermediate data

Size of computations — Some reference points

- DLOG GF(p) 160-digits (Kleinjung 2007): 3.5 + 14 CPU.years
- RSA-768 (Kleinjung et al. 2009): 1500 + 155 CPU.years
- RSA-200 (Bahr, Boem, Franken Kleinjung 2005): 55 + 20 CPU.years
- ECC-2K130 (Bernstein et al.): $\approx$ 16 000 CPU.years
- 10 trillion digits of π (Yee, Kondo 2011) : 12 cores, 90 days: 3 CPU.years
- Largest project in last PRACE call (climate simulation): 16 500 CPU.years

Example 1: EC Point counting (1998)

Example 1: EC Point counting (1998)

- Starting point Lercier PhD (1997)

Example 1: EC Point counting (1998)

- Starting point Lercier PhD (1997)
- Classical computation with 2 phases
  - Phase 1: Compute modular partial information
  - Phase 2: Paste together using collisions search

Example 1: EC Point counting (1998)

- Starting point Lercier PhD (1997)
- Classical computation with 2 phases
  - Phase 1: Compute modular partial information
  - Phase 2: Paste together using collisions search
- Modular data available

Example 1: EC Point counting (1998)

- Starting point Lercier PhD (1997)
- Classical computation with 2 phases
  - Phase 1: Compute modular partial information
  - Phase 2: Paste together using collisions search
- Modular data available
- Classical match-and-sort required about 1 month
  $\Rightarrow$ Power shutdown after 3 weeks !

Example 1: EC Point counting (1998)

- Starting point Lercier PhD (1997)
- Classical computation with 2 phases
  - Phase 1: Compute modular partial information
  - Phase 2: Paste together using collisions search
- Modular data available
- Classical match-and-sort required about 1 month
  $\Rightarrow$ Power shutdown after 3 weeks !
- Back to the drawing board:
  $\Rightarrow$ Chinese and Match, 4 CPUs during a single night

Example 1: EC Point counting (1998)

- Starting point Lercier PhD (1997)
- Classical computation with 2 phases
  - Phase 1: Compute modular partial information
  - Phase 2: Paste together using collisions search
- Modular data available
- Classical match-and-sort required about 1 month
  $\Rightarrow$ Power shutdown after 3 weeks !
- Back to the drawing board:
  $\Rightarrow$ Chinese and Match, 4 CPUs during a single night
- *"Chinese & Match", an alternative to Atkin's "Match and Sort" method used in the SEA algorithm*, Lercier, J. (1999)

Example 1: EC Point counting (1998)

- Starting point Lercier PhD (1997)
- Classical computation with 2 phases
  - Phase 1: Compute modular partial information
  - Phase 2: Paste together using collisions search
- Modular data available
- Classical match-and-sort required about 1 month
  $\Rightarrow$ Power shutdown after 3 weeks !
- Back to the drawing board:
  $\Rightarrow$ Chinese and Match, 4 CPUs during a single night
- *"Chinese & Match", an alternative to Atkin's "Match and Sort" method used in the SEA algorithm*, Lercier, J. (1999)
- Main gain: Reduced memory cost

Example 2: SHA-0 collision (2004)

Example 2: SHA-0 collision (2004)

- Improved version of SHA-0 analysis

Example 2: SHA-0 collision (2004)

- Improved version of SHA-0 analysis
- 4 blocks collision
  $\Rightarrow$ Four consecutive “brute force” steps

Example 2: SHA-0 collision (2004)

- Improved version of SHA-0 analysis
- 4 blocks collision
  $\Rightarrow$ Four consecutive “brute force” steps
- Collision found in 80 000 CPU.hours
  About 9 CPU.years (Three weeks real time on 160 CPUs)

Example 2: SHA-0 collision (2004)

- Improved version of SHA-0 analysis
- 4 blocks collision
  $\Rightarrow$ Four consecutive “brute force” steps
- Collision found in 80 000 CPU.hours
  About 9 CPU.years (Three weeks real time on 160 CPUs)
- Published in *Collisions of SHA-0 and Reduced SHA-1*, Biham, Chen, J., Carribault, Lemuet, Jalby (2005)

Example 3: Triple collisions (2009)

- *Improved generic algorithms for 3-collisions*, Lucks, J. Asiacrypt 2009

Example 3: Triple collisions (2009)

- *Improved generic algorithms for 3-collisions*, Lucks, J. Asiacrypt 2009
- Simple computation with 3 phases
  - Phase 1: Compute iterations $F^i(R)$ from random R
    $\Rightarrow$ Stop at distinguished point
  - Phase 2: Sort by end point values
  - Phase 3: Restart from triples with same end points and recompute

Example 3: Triple collisions (2009)

- *Improved generic algorithms for 3-collisions*, Lucks, J. Asiacrypt 2009
- Simple computation with 3 phases
  - Phase 1: Compute iterations $F^i(R)$ from random R $\Rightarrow$ Stop at distinguished point
  - Phase 2: Sort by end point values
  - Phase 3: Restart from triples with same end points and recompute
- Needs raw computing power, low communication/disk $\Rightarrow$ Phase 1 on CUDA graphics card ($\approx$ 8 times faster than the CPUs on the available machines)

Example 3: Triple collisions (2009)

- *Improved generic algorithms for 3-collisions*, Lucks, J. Asiacrypt 2009
- Simple computation with 3 phases
  - Phase 1: Compute iterations $F^i(R)$ from random R
    $\Rightarrow$ Stop at distinguished point
  - Phase 2: Sort by end point values
  - Phase 3: Restart from triples with same end points and recompute
- Needs raw computing power, low communication/disk
  $\Rightarrow$ Phase 1 on CUDA graphics card ($\approx$ 8 times faster than the CPUs on the available machines)
- Phase 2, easy step, on single CPU

Example 3: Triple collisions (2009)

- *Improved generic algorithms for 3-collisions*, Lucks, J. Asiacrypt 2009
- Simple computation with 3 phases
  - Phase 1: Compute iterations $F^i(R)$ from random R
    $\Rightarrow$ Stop at distinguished point
  - Phase 2: Sort by end point values
  - Phase 3: Restart from triples with same end points and recompute
- Needs raw computing power, low communication/disk
  $\Rightarrow$ Phase 1 on CUDA graphics card ($\approx$ 8 times faster than the CPUs on the available machines)
- Phase 2, easy step, on single CPU
- Phase 3, less costly than Phase 1, harder to code
  Done on CPUs

Example 3: Triple collisions (2009)

- *Improved generic algorithms for 3-collisions*, Lucks, J. Asiacrypt 2009
- Simple computation with 3 phases
  - Phase 1: Compute iterations $F^i(R)$ from random R
    $\Rightarrow$ Stop at distinguished point
  - Phase 2: Sort by end point values
  - Phase 3: Restart from triples with same end points and recompute
- Needs raw computing power, low communication/disk
  $\Rightarrow$ Phase 1 on CUDA graphics card ($\approx$ 8 times faster than the CPUs on the available machines)
- Phase 2, easy step, on single CPU
- Phase 3, less costly than Phase 1, harder to code
  Done on CPUs

- Triple collision on 64-bits cryptographic function
  Magnitude of computation : 100 CPU.days

Example 4: Index calculus

Example 4: Index calculus

- A known landscape:
  - Discrete log. in GF(p): 90 digits (1998), 100 digits (1999), 110 digits (2001) , 120 digits (2001), 130 digits (2005)
  - Discrete log. in GF(2^n): 521 bits (2001), 607 bits (Thomé 2002, 2005) , 613 bits (2005)
  - Discrete log. in GF(p^n): 65537^{25}, 120 digits (2005), 370801^{30}, 168 digits (2005)
  - *When e-th roots become easier than Factoring*, J., Naccache, Thomé 2007
  - Oracle assisted static DH, J., Lercier, Naccache, Thomé 2008
  - Oracle assisted static DH on Oakley curve (Granger, J., Vitse 2010)

Example 4: Index calculus

- A known landscape:
  - Discrete log. in GF(p): 90 digits (1998), 100 digits (1999), 110 digits (2001) , 120 digits (2001), 130 digits (2005)
  - Discrete log. in GF(2^n): 521 bits (2001), 607 bits (Thomé 2002, 2005) , 613 bits (2005)
  - Discrete log. in GF(p^n): 65537^{25}, 120 digits (2005), 370801^{30}, 168 digits (2005)
  - *When e-th roots become easier than Factoring*, J., Naccache, Thomé 2007
  - Oracle assisted static DH, J., Lercier, Naccache, Thomé 2008
  - Oracle assisted static DH on Oakley curve (Granger, J., Vitse 2010)
- Not a routine task !

Index calculus in finite fields

GF(p)	90	100	110	120	130
CPU.days	150	260	70	280	340
Computers	$4 \times 1 + 1$	$8 \times 1 + 1$	1×4	1×4	1×16

GF(2^n)	521	607	613
CPU.days	120	560	1100
Computers	1×4	1×16	4×16

Other	65537^{25}	370801^{30}	RSA-155 e-th roots
CPU.days	2	0.5	2
Computers	1	$1 \times 16 + 1 \times 8$	20

Initial view for EC-DLOG on GF(p^6)

Initial view for EC-DLOG on GF(p^6)

- Theory:
  - Phase 1: Sieving
  - Phase 2: Linear algebra
  - Phase 3: Individual logarithms

Initial view for EC-DLOG on GF(p^6)

- Theory:
  - Phase 1: Sieving
  - Phase 2: Linear algebra
  - Phase 3: Individual logarithms
- Practice:
  - Phase 1:
    - 1a: Sieving
    - 1b: Verification of relations (fast)
  - Phase 2: Linear algebra
  - Phase 3: Individual logarithms

Initial view for EC-DLOG on GF(p^6)

- Theory:
  - Phase 1: Sieving
  - Phase 2: Linear algebra
  - Phase 3: Individual logarithms
- Practice:
  - Phase 1:
    - 1a: Sieving
    - 1b: Verification of relations (fast)
  - Phase 2:
    - 2a: Structured Gaussian Elimination (fast)
    - 2b: Lanczos algorithm
    - 2c: Completing the logarithms (fast)
  - Phase 2: Linear algebra
  - Phase 3: Individual logarithms

Initial view for EC-DLOG on GF(p^6)

- Theory:
  - Phase 1: Sieving
  - Phase 2: Linear algebra
  - Phase 3: Individual logarithms
- Practice:
  - Phase 1:
    - 1a: Sieving
    - 1b: Verification of relations (fast)
  - Phase 2:
    - 2a: Structured Gaussian Elimination (fast)
    - 2b: Lanczos algorithm
    - 2c: Completing the logarithms (fast)
  - Phase 2: Linear algebra
  - Phase 3: Individual logarithms
- View confirmed by 6×22

More data for 6×22

Computation performed on GENCI's Titane computer (Project t2010066445)

More data for 6×22

Computation performed on GENCI's Titane computer (Project t2010066445)

- Sieving: About 1 hour on 200 CPUs

More data for 6×22

Computation performed on GENCI's Titane computer (Project t2010066445)

- Sieving: About 1 hour on 200 CPUs
- SGE: from 50 M eq. in 2.1 M var.
  $\Rightarrow$ 666 K eq./var.

More data for 6×22

Computation performed on GENCI's Titane computer (Project t2010066445)

- Sieving: About 1 hour on 200 CPUs
- SGE: from 50 M eq. in 2.1 M var.
  $\Rightarrow$ 666 K eq./var.
- Lanczos 27 hours on 128 CPUs

More data for 6×22

Computation performed on GENCI's Titane computer (Project t2010066445)

- Sieving: About 1 hour on 200 CPUs
- SGE: from 50 M eq. in 2.1 M var. $\Rightarrow$ 666 K eq./var.
- Lanczos 27 hours on 128 CPUs
- Completion, 10 min single CPU

More data for 6×22

Computation performed on GENCI's Titane computer (Project t2010066445)

- Sieving: About 1 hour on 200 CPUs
- SGE: from 50 M eq. in 2.1 M var.
  $\Rightarrow$ 666 K eq./var.
- Lanczos 27 hours on 128 CPUs
- Completion, 10 min single CPU
- Individual logarithms, a few min, single CPU

More data for 6×22

Computation performed on GENCI's Titane computer (Project t2010066445)

- Sieving: About 1 hour on 200 CPUs
- SGE: from 50 M eq. in 2.1 M var.
  $\Rightarrow$ 666 K eq./var.
- Lanczos 27 hours on 128 CPUs
- Completion, 10 min single CPU
- Individual logarithms, a few min, single CPU

- Total 152 CPU.days

Going to 6×23 and 6×24

Going to 6×23 and 6×24

- 2a: Structured Gaussian Elimination
  - 6×24: Not enough memory. Need to work on disk
  - 6×25: Too slow. Need to multi-thread
  - Corruption of equations on disk:
    $\Rightarrow$ Add a verification of relations

Going to 6×23 and 6×24

- 2a: Structured Gaussian Elimination
  - 6×24: Not enough memory. Need to work on disk
  - 6×25: Too slow. Need to multi-thread
  - Corruption of equations on disk:
    $\Rightarrow$ Add a verification of relations
- 2b: Lanczos: Getting slow
  - Time limit on jobs: need to save/restart
  - Need to supervise the process

More data for 6×23

Computation performed on GENCI's Curie [1]
(PRACE Projects 2010PA0421 and 2011RA0387)

[1] Same computer used for all subsequent computations

More data for 6×23

Computation performed on GENCI's Curie [1]
(PRACE Projects 2010PA0421 and 2011RA0387)

- Sieving: About 3.5 hour on 1024 CPUs

[1] Same computer used for all subsequent computations

More data for 6×23

Computation performed on GENCI's Curie [1]
(PRACE Projects 2010PA0421 and 2011RA0387)

- Sieving: About 3.5 hour on 1024 CPUs
- SGE: Not enough memory
  $\Rightarrow$ Rewrite to work on disk. Becomes too slow: need to multi-thread

[1] Same computer used for all subsequent computations

More data for 6×23

Computation performed on GENCI's Curie [1]
(PRACE Projects 2010PA0421 and 2011RA0387)

- Sieving: About 3.5 hour on 1024 CPUs
- SGE: Not enough memory
  $\Rightarrow$ Rewrite to work on disk. Becomes too slow: need to multi-thread
- New SGE: from 870 Meq. in 4.2 M var.
  $\Rightarrow$ 1 M. eq./var. Using a few hours on 32 CPUs.

[1] Same computer used for all subsequent computations

More data for 6×23

Computation performed on GENCI's Curie [1]
(PRACE Projects 2010PA0421 and 2011RA0387)

- Sieving: About 3.5 hour on 1024 CPUs
- SGE: Not enough memory
  $\Rightarrow$ Rewrite to work on disk. Becomes too slow: need to multi-thread
- New SGE: from 870 Meq. in 4.2 M var.
  $\Rightarrow$ 1 M. eq./var. Using a few hours on 32 CPUs.
- Corruption of some equations on disk:
  $\Rightarrow$ Add a verification of relations

[1] Same computer used for all subsequent computations

More data for 6×23

Computation performed on GENCI's Curie [1]
(PRACE Projects 2010PA0421 and 2011RA0387)

- Sieving: About 3.5 hour on 1024 CPUs
- SGE: Not enough memory
  $\Rightarrow$ Rewrite to work on disk. Becomes too slow: need to multi-thread
- New SGE: from 870 Meq. in 4.2 M var.
  $\Rightarrow$ 1 M. eq./var. Using a few hours on 32 CPUs.
- Corruption of some equations on disk:
  $\Rightarrow$ Add a verification of relations
- Lanczos 73 hours on 64 CPUs

[1] Same computer used for all subsequent computations

More data for 6×23

Computation performed on GENCI's Curie [1]
(PRACE Projects 2010PA0421 and 2011RA0387)

- Sieving: About 3.5 hour on 1024 CPUs
- SGE: Not enough memory
  $\Rightarrow$ Rewrite to work on disk. Becomes too slow: need to multi-thread
- New SGE: from 870 Meq. in 4.2 M var.
  $\Rightarrow$ 1 M. eq./var. Using a few hours on 32 CPUs.
- Corruption of some equations on disk:
  $\Rightarrow$ Add a verification of relations
- Lanczos 73 hours on 64 CPUs
- Completion, 17.5 hours single CPU

[1] Same computer used for all subsequent computations

More data for 6×23

Computation performed on GENCI's Curie [1]
(PRACE Projects 2010PA0421 and 2011RA0387)

- Sieving: About 3.5 hour on 1024 CPUs
- SGE: Not enough memory
  $\Rightarrow$ Rewrite to work on disk. Becomes too slow: need to multi-thread
- New SGE: from 870 Meq. in 4.2 M var.
  $\Rightarrow$ 1 M. eq./var. Using a few hours on 32 CPUs.
- Corruption of some equations on disk:
  $\Rightarrow$ Add a verification of relations
- Lanczos 73 hours on 64 CPUs
- Completion, 17.5 hours single CPU
- Individual logarithms, a few min, single CPU

[1] Same computer used for all subsequent computations

More data for 6×23

Computation performed on GENCI's Curie [1]
(PRACE Projects 2010PA0421 and 2011RA0387)

- Sieving: About 3.5 hour on 1024 CPUs
- SGE: Not enough memory
  $\Rightarrow$ Rewrite to work on disk. Becomes too slow: need to multi-thread
- New SGE: from 870 Meq. in 4.2 M var.
  $\Rightarrow$ 1 M. eq./var. Using a few hours on 32 CPUs.
- Corruption of some equations on disk:
  $\Rightarrow$ Add a verification of relations
- Lanczos 73 hours on 64 CPUs
- Completion, 17.5 hours single CPU
- Individual logarithms, a few min, single CPU

- Total 350 CPU.days

[1] Same computer used for all subsequent computations

More data for 6×24

More data for 6×24

- Sieving: About 15 hours on 1024 CPUs

More data for 6×24

- Sieving: About 15 hours on 1024 CPUs
- New SGE: from 3.5 Geq. in 8.4 M var.
  $\Rightarrow$ 1.7 M. eq./var. Using a few hours on 32 CPUs.

More data for 6×24

- Sieving: About 15 hours on 1024 CPUs
- New SGE: from 3.5 Geq. in 8.4 M var.
  $\Rightarrow$ 1.7 M. eq./var. Using a few hours on 32 CPUs.
- Corruption of some equations on disk:
  $\Rightarrow$ Add a verification of relations

More data for 6×24

- Sieving: About 15 hours on 1024 CPUs
- New SGE: from 3.5 Geq. in 8.4 M var.
  $\Rightarrow$ 1.7 M. eq./var. Using a few hours on 32 CPUs.
- Corruption of some equations on disk:
  $\Rightarrow$ Add a verification of relations
- Lanczos 11 days on 64 CPUs

More data for 6×24

- Sieving: About 15 hours on 1024 CPUs
- New SGE: from 3.5 Geq. in 8.4 M var.
  $\Rightarrow$ 1.7 M. eq./var. Using a few hours on 32 CPUs.
- Corruption of some equations on disk:
  $\Rightarrow$ Add a verification of relations
- Lanczos 11 days on 64 CPUs
- Completion, 13 hours single CPU

More data for 6×24

- Sieving: About 15 hours on 1024 CPUs
- New SGE: from 3.5 Geq. in 8.4 M var.
  $\Rightarrow$ 1.7 M. eq./var. Using a few hours on 32 CPUs.
- Corruption of some equations on disk:
  $\Rightarrow$ Add a verification of relations
- Lanczos 11 days on 64 CPUs
- Completion, 13 hours single CPU
- Individual logarithms, a few min, single CPU

More data for 6×24

- Sieving: About 15 hours on 1024 CPUs
- New SGE: from 3.5 Geq. in 8.4 M var.
  $\Rightarrow$ 1.7 M. eq./var. Using a few hours on 32 CPUs.
- Corruption of some equations on disk:
  $\Rightarrow$ Add a verification of relations
- Lanczos 11 days on 64 CPUs
- Completion, 13 hours single CPU
- Individual logarithms, a few min, single CPU

- Total 1350 CPU.days $\approx$ 3.7 CPU.years

Going to 6×25

Going to 6×25

- Lanczos: Getting slow
  - Time limit on jobs: need to automate save/restart
  - Need to supervise the process

Going to 6×25

- Lanczos: Getting slow
  - Time limit on jobs: need to automate save/restart
  - Need to supervise the process
- Completion of logarithms
  - Related to SGE: Becoming harder
  - Occasional corruption of logarithms on disk !
    $\Rightarrow$ Add a correction step to remove false logs

More data for 6×25

More data for 6×25

- Sieving: About 62 hours on 1024 CPUs

More data for 6×25

- Sieving: About 62 hours on 1024 CPUs
- New SGE: from 14 Geq. in 16.8 M var.
  $\Rightarrow$ 3.1 M. eq. Using a few runs on 32 CPUs. Total 25.5h on 32 CPUs.

More data for 6×25

- Sieving: About 62 hours on 1024 CPUs
- New SGE: from 14 Geq. in 16.8 M var.
  $\Rightarrow$ 3.1 M. eq. Using a few runs on 32 CPUs. Total 25.5h on 32 CPUs.
- Lanczos 28.5 days on 64 CPUs

More data for 6×25

- Sieving: About 62 hours on 1024 CPUs
- New SGE: from 14 Geq. in 16.8 M var.
  $\Rightarrow$ 3.1 M. eq. Using a few runs on 32 CPUs. Total 25.5h on 32 CPUs.
- Lanczos 28.5 days on 64 CPUs
- Completion becoming too slow: multi-threaded version
  $\Rightarrow$ 12 hours on 32 CPUs

More data for 6×25

- Sieving: About 62 hours on 1024 CPUs
- New SGE: from 14 Geq. in 16.8 M var.
  $\Rightarrow$ 3.1 M. eq. Using a few runs on 32 CPUs. Total 25.5h on 32 CPUs.
- Lanczos 28.5 days on 64 CPUs
- Completion becoming too slow: multi-threaded version
  $\Rightarrow$ 12 hours on 32 CPUs
- Individual logarithms, improved code: 1 min, single CPU

More data for 6×25

- Sieving: About 62 hours on 1024 CPUs
- New SGE: from 14 Geq. in 16.8 M var.
  $\Rightarrow$ 3.1 M. eq. Using a few runs on 32 CPUs. Total 25.5h on 32 CPUs.
- Lanczos 28.5 days on 64 CPUs
- Completion becoming too slow: multi-threaded version
  $\Rightarrow$ 12 hours on 32 CPUs
- Individual logarithms, improved code: 1 min, single CPU

- Total 4470 CPU.days $\approx$ 12 CPU.years

EC-DLOG on GF(p^6): toward 6×26

EC-DLOG on GF(p^6): toward 6×26

- Theory:
  - Phase 1: Sieving
  - Phase 2: Linear algebra
  - Phase 3: Individual logarithms

EC-DLOG on GF(p^6): toward 6×26

- Theory:
  - Phase 1: Sieving
  - Phase 2: Linear algebra
  - Phase 3: Individual logarithms
- Practice:
  - Phase 1:
    - 1a: Sieving
    - 1b: Verification of relations (fast)

EC-DLOG on GF(p^6): toward 6×26

- Theory:
  - Phase 1: Sieving
  - Phase 2: Linear algebra
  - Phase 3: Individual logarithms
- Practice:
  - Phase 1:
    - 1a: Sieving
    - 1b: Verification of relations (fast)
  - Phase 2:
    - 2a: Structured Gaussian Elimination
    - 2b: Verification of relations
    - 2c: Lanczos algorithm (About 4 months expected)
    - 2d: Completing/Correcting the logarithms

EC-DLOG on GF(p^6): toward 6×26

- Theory:
  - Phase 1: Sieving
  - Phase 2: Linear algebra
  - Phase 3: Individual logarithms
- Practice:
  - Phase 1:
    - 1a: Sieving
    - 1b: Verification of relations (fast)
  - Phase 2:
    - 2a: Structured Gaussian Elimination
    - 2b: Verification of relations
    - 2c: Lanczos algorithm (About 4 months expected)
    - 2d: Completing/Correcting the logarithms
  - Phase 3: Individual logarithms (fast)
- New view confirmed by 6×25

Toward 6×26

- Sieving and verification OK
  8192 CPUs for 24 hours
- SGE OK: From 40 Geq in 33.5 M var
  $\Rightarrow$ 5.9 M eq. A few 10h runs on 32 CPUs
- Lanczos expected to 4 months on 64 CPUs:

Toward 6×26

- Sieving and verification OK
  8192 CPUs for 24 hours
- SGE OK: From 40 Geq in 33.5 M var
  $\Rightarrow$ 5.9 M eq. A few 10h runs on 32 CPUs
- Lanczos expected to 4 months on 64 CPUs:
  - Started on Sept. 22^{nb}.

Toward 6×26

- Sieving and verification OK
  8192 CPUs for 24 hours
- SGE OK: From 40 Geq in 33.5 M var
  $\Rightarrow$ 5.9 M eq. A few 10h runs on 32 CPUs
- Lanczos expected to 4 months on 64 CPUs:
  - Started on Sept. 22nb.
  - Slower than expected in real time
    Machine busy, need to wait between runs

Toward 6×26

- Sieving and verification OK
  8192 CPUs for 24 hours
- SGE OK: From 40 Geq in 33.5 M var
  $\Rightarrow$ 5.9 M eq. A few 10h runs on 32 CPUs
- Lanczos expected to 4 months on 64 CPUs:
  - Started on Sept. 22^{nb}.
  - Slower than expected in real time
    Machine busy, need to wait between runs
  - End expected on Feb. 4^{th}

Toward 6×26

- Sieving and verification OK
  8192 CPUs for 24 hours
- SGE OK: From 40 Geq in 33.5 M var
  $\Rightarrow$ 5.9 M eq. A few 10h runs on 32 CPUs
- Lanczos expected to 4 months on 64 CPUs:
  - Started on Sept. 22^{nb}.
  - Slower than expected in real time
    Machine busy, need to wait between runs
  - End expected on Feb. 4^{th}
  - Orthogonalization did not stop !

Toward 6×26

- Sieving and verification OK
  8192 CPUs for 24 hours
- SGE OK: From 40 Geq in 33.5 M var
  $\Rightarrow$ 5.9 M eq. A few 10h runs on 32 CPUs
- Lanczos expected to 4 months on 64 CPUs:
  - Started on Sept. 22^{nb}.
  - Slower than expected in real time
    Machine busy, need to wait between runs
  - End expected on Feb. 4^{th}
  - Orthogonalization did not stop !
- Failure: how to proceed ?

Toward 6×26

- Sieving and verification OK
  8192 CPUs for 24 hours
- SGE OK: From 40 Geq in 33.5 M var
  $\Rightarrow$ 5.9 M eq. A few 10h runs on 32 CPUs
- Lanczos expected to 4 months on 64 CPUs:
  - Started on Sept. 22^{nb}.
  - Slower than expected in real time
    Machine busy, need to wait between runs
  - End expected on Feb. 4th
  - Orthogonalization did not stop !
- Failure: how to proceed ?
  - Option 1: Add a sanity check and restart

Toward 6×26

- Sieving and verification OK
  8192 CPUs for 24 hours
- SGE OK: From 40 Geq in 33.5 M var
  $\Rightarrow$ 5.9 M eq. A few 10h runs on 32 CPUs
- Lanczos expected to 4 months on 64 CPUs:
  - Started on Sept. 22^{nb}.
  - Slower than expected in real time
    Machine busy, need to wait between runs
  - End expected on Feb. 4th
  - Orthogonalization did not stop !
- Failure: how to proceed ?
  - Option 1: Add a sanity check and restart
  - Option 2: Improve Lanczos for more CPUs

Toward 6×26

- Sieving and verification OK
  8192 CPUs for 24 hours
- SGE OK: From 40 Geq in 33.5 M var
  $\Rightarrow$ 5.9 M eq. A few 10h runs on 32 CPUs
- Lanczos expected to 4 months on 64 CPUs:
  - Started on Sept. 22^{nb}.
  - Slower than expected in real time
    Machine busy, need to wait between runs
  - End expected on Feb. 4^{th}
  - Orthogonalization did not stop !
- Failure: how to proceed ?
  - Option 1: Add a sanity check and restart
  - Option 2: Improve Lanczos for more CPUs
  - Option 3: Back to the drawing board

Back to the drawing board

Back to the drawing board

- Solution known: Block Wiedemann (Coppersmith)

Back to the drawing board

- Solution known: Block Wiedemann (Coppersmith)
  - Used by Thomé for $GF(2^{603})$. 480 K eqs.
    Need 4 weeks on 6 quadri-CPUs computers.

Back to the drawing board

- Solution known: Block Wiedemann (Coppersmith)
  - Used by Thomé for GF(2^{603}). 480 K eqs.
    Need 4 weeks on 6 quadri-CPUs computers.
  - Used by Kleinjung for GF(p), 160-digits, 2.2 Meqs
    8 jobs (12-24 CPUs) each, 14 CPU.years (at least 4 weeks)

Back to the drawing board

- Solution known: Block Wiedemann (Coppersmith)
  - Used by Thomé for GF(2^{603}). 480 K eqs.
    Need 4 weeks on 6 quadri-CPUs computers.
  - Used by Kleinjung for GF(p), 160-digits, 2.2 Meqs
    8 jobs (12-24 CPUs) each, 14 CPU.years (at least 4 weeks)
- Three Phases:

Back to the drawing board

- Solution known: Block Wiedemann (Coppersmith)
  - Used by Thomé for GF(2^{603}). 480 K eqs.
    Need 4 weeks on 6 quadri-CPUs computers.
  - Used by Kleinjung for GF(p), 160-digits, 2.2 Meqs
    8 jobs (12-24 CPUs) each, 14 CPU.years (at least 4 weeks)
- Three Phases:
  - Several iterated matrix multiplications in parallel

Back to the drawing board

- Solution known: Block Wiedemann (Coppersmith)
  - Used by Thomé for GF(2^{603}). 480 K eqs.
    Need 4 weeks on 6 quadri-CPUs computers.
  - Used by Kleinjung for GF(p), 160-digits, 2.2 Meqs
    8 jobs (12-24 CPUs) each, 14 CPU.years (at least 4 weeks)
- Three Phases:
  - Several iterated matrix multiplications in parallel
  - Find linear relation in sequence:
    Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm, Thomé (2001/2002)

Back to the drawing board

- Solution known: Block Wiedemann (Coppersmith)
  - Used by Thomé for $GF(2^{603})$. 480 K eqs.
    Need 4 weeks on 6 quadri-CPUs computers.
  - Used by Kleinjung for GF(p), 160-digits, 2.2 Meqs
    8 jobs (12-24 CPUs) each, 14 CPU.years (at least 4 weeks)
- Three Phases:
  - Several iterated matrix multiplications in parallel
  - Find linear relation in sequence:
    Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm, Thomé (2001/2002)
  - Recompute iterated matrix multiplications in parallel to obtain solution

Back to the drawing board

- Solution known: Block Wiedemann (Coppersmith)
  - Used by Thomé for GF(2^{603}). 480 K eqs.
    Need 4 weeks on 6 quadri-CPUs computers.
  - Used by Kleinjung for GF(p), 160-digits, 2.2 Meqs
    8 jobs (12-24 CPUs) each, 14 CPU.years (at least 4 weeks)
- Three Phases:
  - Several iterated matrix multiplications in parallel
  - Find linear relation in sequence:
    Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm, Thomé (2001/2002)
  - Recompute iterated matrix multiplications in parallel to obtain solution
- Need to scale up the approach

New Linear Algebra, testing on 6×25

- Lanczos on 64 cores
- Lanczos Total CPU time $\approx$ 43 800 hours
- Lanczos Real time (without waits) $\approx$ 28.5 days

New Linear Algebra, testing on 6×25

- Lanczos on 64 cores
- Lanczos Total CPU time $\approx 43\,800$ hours
- Lanczos Real time (without waits) ≈ 28.5 days

- First Matrix Vector Phase: $\approx$ 33h30 on 1024 cores
  32 independent sequences

New Linear Algebra, testing on 6×25

- Lanczos on 64 cores
- Lanczos Total CPU time $\approx$ 43 800 hours
- Lanczos Real time (without waits) $\approx$ 28.5 days

- First Matrix Vector Phase: $\approx$ 33h30 on 1024 cores
  32 independent sequences
- Thomé's algorithm: $\approx$ 9h30 on 32 cores

New Linear Algebra, testing on 6×25

- Lanczos on 64 cores
- Lanczos Total CPU time $\approx$ 43 800 hours
- Lanczos Real time (without waits) $\approx$ 28.5 days

- First Matrix Vector Phase: $\approx$ 33h30 on 1024 cores
  32 independent sequences
- Thomé's algorithm: $\approx$ 9h30 on 32 cores
- Second Matrix Vector Phase: $\approx$ 15h30 on 1024 cores

New Linear Algebra, testing on 6×25

- Lanczos on 64 cores
- Lanczos Total CPU time $\approx$ 43 800 hours
- Lanczos Real time (without waits) $\approx$ 28.5 days

- First Matrix Vector Phase: $\approx$ 33h30 on 1024 cores
  32 independent sequences
- Thomé's algorithm: $\approx$ 9h30 on 32 cores
- Second Matrix Vector Phase: $\approx$ 15h30 on 1024 cores
- Total CPU time $\approx$ 50 500 hours, 2100 CPU.days
- Real time (without waits) $\approx$ 2.5 days

New Linear Algebra, testing on 6×25

- Lanczos on 64 cores
- Lanczos Total CPU time $\approx$ 43 800 hours
- Lanczos Real time (without waits) $\approx$ 28.5 days

- First Matrix Vector Phase: $\approx$ 33h30 on 1024 cores
  32 independent sequences
- Thomé's algorithm: $\approx$ 9h30 on 32 cores
- Second Matrix Vector Phase: $\approx$ 15h30 on 1024 cores
- Total CPU time $\approx$ 50 500 hours, 2100 CPU.days
- Real time (without waits) $\approx$ 2.5 days
- New total real time including Sieving: $\approx$ 5 days
  $\approx$ 14 CPU.years

New linear algebra 6×26 ?

New linear algebra 6×26 ?

- First Matrix Vector Phase: $\approx$ 125 h on 1024 cores
  32 independent sequences

New linear algebra 6×26 ?

- First Matrix Vector Phase: $\approx$ 125 h on 1024 cores
  32 independent sequences
- Started March 28$^{\text{th}}$

New linear algebra 6×26 ?

- First Matrix Vector Phase: $\approx$ 125 h on 1024 cores
  32 independent sequences
- Started March 28th

```
Due to an electrical problem, CURIE is
unavailable since the 3th april 2012 at
8:30pm.
```

New linear algebra 6×26 ?

- First Matrix Vector Phase: $\approx$ 125 h on 1024 cores
  32 independent sequences
- Started March 28^{th}

> `Due to an electrical problem, CURIE is unavailable since the 3th april 2012 at 8:30pm.`

> `General power cut on high voltage line is solved.  The TGCC center is operational and CURIE is now available.  (April 4th, 17:30)`

New linear algebra 6×26 ?

- First Matrix Vector Phase: $\approx$ 125 h on 1024 cores
  32 independent sequences
- Started March 28^{th}

  > `Due to an electrical problem, CURIE is unavailable since the 3th april 2012 at 8:30pm.`

  > `General power cut on high voltage line is solved. The TGCC center is operational and CURIE is now available. (April 4th, 17:30)`

- Still running . . . (Curie very busy these days)

Conclusion

Questions ?