ECM at Work

Joppe W. Bos[1] and Thorsten Kleinjung[2]

[1] Microsoft Research, Redmond, USA

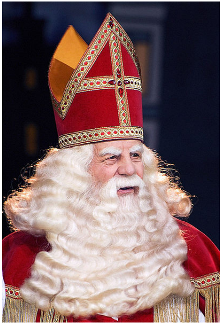
[2] Laboratory for Cryptologic Algorithms, EPFL, Lausanne, Switzerland

ECM at Work

Joppe W. Bos[1] and Thorsten Kleinjung[2]

[1] Microsoft Research, Redmond, USA

[2] Laboratory for Cryptologic Algorithms, EPFL, Lausanne, Switzerland



Happy Sinterklaas!

Microsoft Research

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Security assessment of public-key cryptography

Most-widely used public-key cryptosystem: RSA
Integer factorization problem ($n = pq$ with $p \approx q$)
Factoring RSA-like numbers: Number Field Sieve (NFS)

Security assessment of public-key cryptography

Most-widely used public-key cryptosystem: RSA
Integer factorization problem ($n = pq$ with $p \approx q$)
Factoring RSA-like numbers: Number Field Sieve (NFS)

Inside NFS factor many small numbers: cofactorization

$$\left.\begin{array}{c}\text{primality testing}\\ \text{trial division}\\ p-1,\ \text{QS}\\ \text{ECM}\end{array}\right\} \approx 1/3 \text{ of the run-time for RSA-768 [CRYPTO'10]}$$

A. K. Lenstra and H. W. Lenstra, Jr. The Development of the Number Field Sieve, Lecture Notes in Mathematics, 1993
H. W. Lenstra Jr. Factoring integers with elliptic curves. Annals of Mathematics, 1987.

Security assessment of public-key cryptography

Most-widely used public-key cryptosystem: RSA
Integer factorization problem ($n = pq$ with $p \approx q$)
Factoring RSA-like numbers: Number Field Sieve (NFS)

Inside NFS factor many small numbers: cofactorization

$$\left.\begin{array}{c}\text{primality testing}\\ \text{trial division}\\ p-1,\ \text{QS}\\ \underbrace{\textbf{ECM}}\end{array}\right\} \approx 1/3 \text{ of the run-time for RSA-768 [CRYPTO'10]}$$

Offloading this work (to FPGA, GPU) is an active research area,
since faster cofactorization $\rightarrow$ faster NFS

A. K. Lenstra and H. W. Lenstra, Jr. The Development of the Number Field Sieve, Lecture Notes in Mathematics, 1993
H. W. Lenstra Jr. Factoring integers with elliptic curves. Annals of Mathematics, 1987.

Elliptic Curve Method (ECM)

Try and factor $n = p \cdot q$ with $1 < p < q < n$. Repeat:

- Pick a random point P and construct an elliptic E over $\mathbf{Z}/n\mathbf{Z}$ containing P
- Compute $Q = kP \in E(\mathbf{Z}/n\mathbf{Z})$ for some $k \in \mathbf{Z}$
- If $\#E(\mathbf{F}_p) \mid k$ (and $\#E(\mathbf{Z}/q\mathbf{Z}) \nmid k$) then Q and the neutral element become the same modulo p
- $p = \gcd(n, Q_z)$

In practice given a bound $B_1 \in \mathbf{Z}$: $k = \mathrm{lcm}(1, 2, \ldots, B_1)$

H. W. Lenstra Jr. Factoring integers with elliptic curves. Annals of Mathematics, 1987.

Elliptic Curve Method (ECM)

Try and factor $n = p \cdot q$ with $1 < p < q < n$. Repeat:

- Pick a random point P and construct an elliptic E over $\mathbf{Z}/n\mathbf{Z}$ containing P
- Compute $Q = kP \in E(\mathbf{Z}/n\mathbf{Z})$ for some $k \in \mathbf{Z}$
- If $\#E(\mathbf{F}_p) \mid k$ (and $\#E(\mathbf{Z}/q\mathbf{Z}) \nmid k$) then Q and the neutral element become the same modulo p
- $p = \gcd(n, Q_z)$

In practice given a bound $B_1 \in \mathbf{Z}$: $k = \mathrm{lcm}(1, 2, \ldots, B_1)$

$$O(e^{(\sqrt{2}+o(1))(\sqrt{\log p \log \log p})} M(\log n))$$

- $M(\log n)$ represents the complexity of multiplication modulo n
- $o(1)$ is for $p \to \infty$

H. W. Lenstra Jr. Factoring integers with elliptic curves. Annals of Mathematics, 1987.

Edwards Curves (based on work by Euler & Gauss)

- Edwards curves
- Twisted Edwards curves
- Inverted Edwards coordinates
- Extended twisted Edwards coordinates

A twisted Edwards curve is defined $(ad(a-d) \neq 0)$

$$ax^2 + y^2 = 1 + dx^2y^2 \text{ and } (ax^2 + y^2)z^2 = z^4 + dx^2y^2$$

2007: H. M. Edwards. A normal form for elliptic curves. Bulletin of the American Mathematical Society
2007: D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. Asiacrypt
2008: H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson. Twisted Edwards curves revisited. Asiacrypt

Edwards Curves (based on work by Euler & Gauss)

- Edwards curves
- Twisted Edwards curves
- Inverted Edwards coordinates
- Extended twisted Edwards coordinates

A twisted Edwards curve is defined $(ad(a-d) \neq 0)$

$$ax^2 + y^2 = 1 + dx^2y^2 \text{ and } (ax^2 + y^2)z^2 = z^4 + dx^2y^2$$

Elliptic Curve Point Addition $\begin{cases} a = -1\text{: } 8M \\ a = -1, z_1 = 1\text{: } 7M \end{cases}$

Elliptic Curve Point Duplication: $a = -1$: 3M + 4S

2007: H. M. Edwards. A normal form for elliptic curves. Bulletin of the American Mathematical Society
2007: D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. Asiacrypt
2008: H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson. Twisted Edwards curves revisited. Asiacrypt

Motivation

B1	GMP-ECM Montgomery curves				EECM-MPFQ Edwards curves			
	#**S**	#**M**	#**S**+#**M**	#R	#**S**	#**M**	#**S**+#**M**	#R
256	1 066	2 025	3 091	14	1 436	1 638	3 074	38
512	2 200	4 210	6 410	14	2 952	3 183	6 135	62
1024	4 422	8 494	12 916	14	5 892	6 144	12 036	134
8192	35 508	68 920	104 428	14	47 156	45 884	93 040	550

P. Zimmermann and B. Dodson. 20 Years of ECM. Algorithmic Number Theory Symposium – ANTS 2006
D. J. Bernstein, P. Birkner, T. Lange, and C. Peters. ECM using Edwards curves. Mathematics of Computation (to appear)
D. J. Bernstein, P. Birkner, and T. Lange. Starfish on strike. Latincrypt, 2010

Motivation

B1	GMP-ECM Montgomery curves				EECM-MPFQ Edwards curves			
	#**S**	#**M**	#**S**+#**M**	#*R*	#**S**	#**M**	#**S**+#**M**	#*R*
256	1 066	2 025	3 091	14	1 436	1 638	3 074	38
512	2 200	4 210	6 410	14	2 952	3 183	6 135	62
1024	4 422	8 494	12 916	14	5 892	6 144	12 036	134
8192	35 508	68 920	104 428	14	47 156	45 884	93 040	550

Edwards curves vs Montgomery curves

- faster EC-arithmetic
- more memory is required

$\Rightarrow$ Difficult to run Edwards-ECM fast on memory-constrained devices

This work: **faster**, *memory* **efficient** Edwards ECM (on GPUs)

P. Zimmermann and B. Dodson. 20 Years of ECM. Algorithmic Number Theory Symposium – ANTS 2006
D. J. Bernstein, P. Birkner, T. Lange, and C. Peters. ECM using Edwards curves. Mathematics of Computation (to appear)
D. J. Bernstein, P. Birkner, and T. Lange. Starfish on strike. Latincrypt, 2010

Elliptic Curve *Constant* Scalar Multiplication

In practice people use the same B_1 for many numbers:
Can we do better for a fixed B_1?

Elliptic Curve *Constant* Scalar Multiplication

In practice people use the same B_1 for many numbers:
Can we do better for a fixed B_1?

B. Dixon and A. K. Lenstra. Massively parallel elliptic curve factoring. Eurocrypt 1992.

Recall: $k = \mathrm{lcm}(1, 2, \ldots, B_1) = \prod_i p_i$ with $p_i \leq B_1$ prime.

- **Observation:** Use double-and-add approach, no additional storage.
  Low Hamming-weight integers $\rightarrow$ fewer EC-additions
- **Idea:** Search for low-weight prime products
  Partition the set of primes in subsets of cardinality of most three
- **Result:** Lowered the weight by $\approx$ a factor three

(Computing the shortest addition chain is *conjectured* to be NP-hard)

Elliptic Curve *Constant* Scalar Multiplication

In practice people use the same B_1 for many numbers:
Can we do better for a fixed B_1?

B. Dixon and A. K. Lenstra. Massively parallel elliptic curve factoring. Eurocrypt 1992.

Recall: $k = \mathrm{lcm}(1, 2, \ldots, B_1) = \prod_i p_i$ with $p_i \leq B_1$ prime.

- **Observation:** Use double-and-add approach, no additional storage.
  Low Hamming-weight integers $\rightarrow$ fewer EC-additions
- **Idea:** Search for low-weight prime products
  Partition the set of primes in subsets of cardinality of most three
- **Result:** Lowered the weight by $\approx$ a factor three

$$1028107 \cdot 1030639 \cdot 1097101 = 1162496086223388673$$
$$w(1028107) = 10, \quad w(1030639) = 16,$$
$$w(1097101) = 11, \quad w(1162496086223388673) = 8$$

(Computing the shortest addition chain is *conjectured* to be NP-hard)

Elliptic Curve *Constant* Scalar Multiplication

We try the opposite approach ($c(s) := \#\mathbf{A}$ in the addition chain)

- Generate integers s with "good" $\mathbf{D}/\mathbf{A}$ ratio
- Test for B_1-smoothness and factor these integers $s = \prod_j \hat{s}_j$

J. Franke, T. Kleinjung, F. Morain, and T. Wirth. Proving the primality of very large numbers with fastECPP. Algorithmic Number Theory 2004

Subset cover problem under minimization constraints

Elliptic Curve *Constant* Scalar Multiplication

We try the opposite approach ($c(s) := \#\mathbf{A}$ in the addition chain)

- Generate integers s with "good" $\mathbf{D}/\mathbf{A}$ ratio
- Test for B_1-smoothness and factor these integers $s = \prod_j \hat{s}_j$

  J. Franke, T. Kleinjung, F. Morain, and T. Wirth. Proving the primality of very large numbers with fastECPP. Algorithmic Number Theory 2004

- Combine integers s_i such that

$$\prod_i s_i = \prod_i \prod_j \hat{s}_{i,j} = k = \operatorname{lcm}(1, \ldots, B_1) = \prod_\ell p_\ell$$

  i.e. all the $\hat{s}_{i,j}$ match all the p_ℓ

- Such that $\sum_i c(s_i = \prod_j \hat{s}_{i,j}) < c'(\prod_\ell p_\ell) = c'(k)$

Subset cover problem under minimization constraints

Addition/subtraction chain resulting in s

$$s = a_r, \ldots, a_1, a_0 = 1$$

s.t. every $a_i = a_j \pm a_k$ with $0 \leq j, k < i$.

	No-storage	Low-Storage
#integers	$\binom{\mathbf{D}-1}{\mathbf{A}-1} \cdot 2^{\mathbf{A}}$	$\binom{\mathbf{D}-1}{\mathbf{A}-1} \cdot \mathbf{A}! \cdot 2^{\mathbf{A}}$

Addition/subtraction chain resulting in s

$$s = a_r, \dots, a_1, a_0 = 1$$

s.t. every $a_i = a_j \pm a_k$ with $0 \le j, k < i$.

	No-storage	Low-Storage
#integers	$\binom{\mathbf{D}-1}{\mathbf{A}-1} \cdot 2^{\mathbf{A}}$	$\binom{\mathbf{D}-1}{\mathbf{A}-1} \cdot \mathbf{A}! \cdot 2^{\mathbf{A}}$

Combining the smooth-integers

- Greedy approach (use good **D**/**A** ratios first)
- Selection process is randomized
- Score according to the size of the prime divisors
- Left-overs are done using brute-force

All technical details in our paper!

$2.9 \cdot 10^9$-smoothness testing

No-storage setting			Low-storage setting		
A	**D**	**#ST**	**A**	**D**	**#ST**
1	$5 - 200$	$3.920 \cdot 10^2$	1	$5 - 250$	$4.920 \cdot 10^2$
2	$10 - 200$	$7.946 \cdot 10^4$	2	$10 - 250$	$2.487 \cdot 10^5$
3	$15 - 200$	$1.050 \cdot 10^7$	3	$15 - 250$	$1.235 \cdot 10^8$
4	$20 - 200$	$1.035 \cdot 10^9$	4	$20 - 250$	$6.101 \cdot 10^{10}$
5	$25 - 200$	$8.114 \cdot 10^{10}$	5	$25 - 158$	$2.956 \cdot 10^{12}$
			5	**159 – 220**	$\mathbf{1.331 \cdot 10^{11}}$
6	$30 - 173$	$2.183 \cdot 10^{12}$	**6**	**60 – 176**	$\mathbf{2.513 \cdot 10^{11}}$
7	$35 - 84$	$5.791 \cdot 10^{11}$			
Total		$2.844 \cdot 10^{12}$			$3.403 \cdot 10^{12}$

$2.9 \cdot 10^9$-smoothness tests on our mini-cluster using 4.5 GB memory
(5×8 Intel Xeon CPU E5430 2.66GHz)
Results obtained in $\approx$ 18 months

Example $B_1 = 256$, No-Storage

#D	#A	product	addition chain
11	1	$89 \cdot 23$	$S_0 D^{11}$
14	2	$197 \cdot 83$	$S_0 D^5 S_0 D^9$
15	2	$193 \cdot 191$	$S_0 D^{12} A_0 D^3$
15	2	$199 \cdot 19 \cdot 13$	$A_0 D^{14} A_0 D^1$
18	1	$109 \cdot 37 \cdot 13 \cdot 5$	$A_0 D^{18}$
19	2	$157 \cdot 53 \cdot 7 \cdot 3 \cdot 3$	$S_0 D^6 S_0 D^{13}$
21	3	$223 \cdot 137 \cdot 103$	$A_0 D^{10} A_0 D^{10} A_0 D^1$
23	3	$179 \cdot 149 \cdot 61 \cdot 5$	$S_0 D^{13} A_0 D^5 S_0 D^5$
28	1	$127 \cdot 113 \cdot 43 \cdot 29 \cdot 5 \cdot 3$	$S_0 D^{28}$
30	3	$181 \cdot 173 \cdot 167 \cdot 11 \cdot 7 \cdot 3$	$A_0 D^{11} A_0 D^{16} A_0 D^3$
33	5	$211 \cdot 73 \cdot 67 \cdot 59 \cdot 47 \cdot 3$	$S_0 D^6 A_0 D^2 A_0 D^{11} S_0 D^3 S_0 D^{11}$
36	4	$241 \cdot 131 \cdot 101 \cdot 79 \cdot 31 \cdot 11$	$A_0 D^2 A_0 D^{16} A_0 D^{16} A_0 D^2$
41	4	$233 \cdot 229 \cdot 163 \cdot 139 \cdot 107 \cdot 17$	$S_0 D^9 S_0 D^4 S_0 D^{11} S_0 D^{17}$
49	5	$251 \cdot 239 \cdot 227 \cdot 151 \cdot 97 \cdot 71 \cdot 41$	$S_0 D^3 S_0 D^{29} A_0 D^4 A_0 D^8 A_0 D^5$
8	0	2^8	D^8
361	38	**Total**	

Speedup

B_1		$\#\mathbf{M} + \#\mathbf{S}$	speedup	$\#R$	reduction
256	[1]	3 074		38	
No-storage		2 844	1.08	10	3.80
Low-storage		2 831	1.09	22	1.73
512	[1]	6 135		62	
No-storage		5 806	1.06	10	6.20
Low-storage		5 740	1.07	22	2.82
1 024	[1]	12 036		134	
No-storage		11 508	1.05	10	13.40
Low-storage		11 375	1.06	22	6.09
8 192	[1]	93 040		550	
No-storage		91 074	1.02	10	55.00
Low-storage		89 991	1.03	22	25.00

[1] Starfish on Strike, D. J. Bernstein, P. Birkner, T. Lange, Latincrypt 2010

Speedup

B_1		$\#\mathbf{M} + \#\mathbf{S}$	speedup	$\#R$	reduction
256	[1]	3 074		38	
No-storage		2 844	1.08	10	3.80
Low-storage		2 831	1.09	22	1.73
512	[1]	6 135		62	
No-storage		5 806	1.06	10	6.20
Low-storage		5 740	1.07	22	2.82
1 024	[1]	12 036		134	
No-storage		11 508	1.05	10	13.40
Low-storage		11 375	1.06	22	6.09
8 192	[1]	93 040		550	
No-storage		91 074	1.02	10	55.00
Low-storage		89 991	1.03	22	25.00

This does not take the memory overhead into account...

We expect a higher speedup in practice!

[1] Starfish on Strike, D. J. Bernstein, P. Birkner, T. Lange, Latincrypt 2010

Performance Comparison, 192-bit moduli

	performance (#curves), $B_1 = 960$ (1/sec)	(1/\$100)	performance ratio
Intel i7 [gnfslinux]	13 661	4 554	
Intel i7 [EECM]	8 677	2 892	
V4SX35-10 [FPL'10]	3 586	766	
V4SX25-10 [FCCM'07]	7 910	2 654	
performance (#curves), $B_1 = 8192$			
GTX 295 [SHARCS'09]	5 895	-	
Intel i7 [gnfslinux]	1 629	543	
Intel i7 [EECM]	1 092	364	

Performance Comparison, 192-bit moduli

performance (#curves), $B_1 = 960$	(1/sec)	(1/\$100)	performance ratio
GTX 580, no-storage	171 486	42 872	1.00
Intel i7 [gnfslinux]	13 661	4 554	9.41
Intel i7 [EECM]	8 677	2 892	14.82
V4SX35-10 [FPL'10]	3 586	766	55.97
V4SX25-10 [FCCM'07]	7 910	2 654	16.15

performance (#curves), $B_1 = 8192$			
GTX 295 [SHARCS'09]	5 895	-	-
GTX 580, no-storage	19 869	4 967	1.00
Intel i7 [gnfslinux]	1 629	543	9.15
Intel i7 [EECM]	1 092	364	13.65

Performance Comparison, 192-bit moduli

	performance (#curves), $B_1 = 960$ (1/sec)	(1/$100)	performance ratio
GTX 580, no-storage	171 486	42 872	1.00
GTX 580, windowing	79 170	19 793	2.17
Intel i7 [gnfslinux]	13 661	4 554	9.41
Intel i7 [EECM]	8 677	2 892	14.82
V4SX35-10 [FPL'10]	3 586	766	55.97
V4SX25-10 [FCCM'07]	7 910	2 654	16.15

	performance (#curves), $B_1 = 8192$		
GTX 295 [SHARCS'09]	5 895	-	-
GTX 580, no-storage	19 869	4 967	1.00
GTX 580, windowing	9 106	2 277	2.18
Intel i7 [gnfslinux]	1 629	543	9.15
Intel i7 [EECM]	1 092	364	13.65

Conclusions

- Methods to precompute *"good"* addition chains
- Speedup elliptic curve scalar multiplication with constants
- Very suitable for parallel architectures
- Can also be used to speed up cryptographic protocols where the scalar is fixed

Compared to the state-of-the-art in cofactorization

- Reduces the memory up to a factor 56
- On GPUs $\rightarrow$ more than a two-fold performance speedup
- New (GPU) Edwards-ECM throughput records

Get the latest addition-chains from:

http://research.microsoft.com/ecmatwork