Attacking Elliptic Curve Challenges

Daniel V. Bailey, Lejla Batina, Daniel J. Bernstein, Peter Birkner, Joppe W. Bos, Hsieh-Chung Chen, Chen-Mou Cheng, Gauthier van Damme, Giacomo de Meulenaer, Luis Julian Dominguez Perez, Junfeng Fan, Tim Güneysu, Frank Gürkaynak, Thorsten Kleinjung, Tanja Lange, Nele Mentens, Ruben Niederhagen, Christof Paar, Francesco Regazzoni, Peter Schwabe, Leif Uhsadel, Anthony Van Herrewege, Bo-Yin Yang,
and several individuals and institutions donating computer time

2010.09.08

The Certicom challenges

1997: Certicom announces several ECDLP prizes:

> *The Challenge is to compute the ECC private keys from the given list of ECC public keys and associated system parameters. This is the type of problem facing an adversary who wishes to completely defeat an elliptic curve cryptosystem.*

Objectives stated by Certicom:

- Increase community's understanding of ECDLP difficulty.
- Confirm theoretical comparisons of ECC and RSA.
- Help users select suitable key sizes.
- Compare ECDLP difficulty for $\mathbf{F}_{2^m}$ and $\mathbf{F}_p$.
- Compare $\mathbf{F}_{2^m}$ ECDLP difficulty for random and Koblitz.
- Stimulate research in algorithmic number theory.

The Certicom challenges, level 0: exercises

Bits	Name	"Estimated number of machine days"	Prize
79	ECCp-79	146	book
79	ECC2-79	352	book
89	ECCp-89	4360	book
89	ECC2-89	11278	book
97	ECC2K-95	8637	$5000
97	ECCp-97	71982	$5000
97	ECC2-97	180448	$5000

Certicom believes that it is feasible that the 79-bit exercises could be solved in a matter of hours, the 89-bit exercises could be solved in a matter of days, and the 97-bit exercises in a matter of weeks using a network of 3000 computers.

The Certicom challenges, level 1

Bits	Name	"Estimated number of machine days"	Prize
109	ECC2K-108	1300000	$10000
109	ECCp-109	9000000	$10000
109	ECC2-109	21000000	$10000
131	ECC2K-130	2700000000	$20000
131	ECCp-131	23000000000	$20000
131	ECC2-131	66000000000	$20000

The 109-bit Level I challenges are feasible using a very large network of computers. The 131-bit Level I challenges are expected to be infeasible against realistic software and hardware attacks, unless of course, a new algorithm for the ECDLP is discovered.

The Certicom challenges, level 2

Bits	Name	"Estimated number of machine days"	Prize
163	ECC2K-163	320000000000000	\$30000
163	ECCp-163	2300000000000000	\$30000
163	ECC2-163	6200000000000000	\$30000
191	ECCp-191	480000000000000000000	\$40000
191	ECC2-191	1000000000000000000000	\$40000
239	ECC2K-238	920000000000000000000000000	\$50000
239	ECCp-239	14000000000000000000000000000	\$50000
239	ECC2-238	21000000000000000000000000000	\$50000
359	ECCp-359	$\approx \infty$	\$100000

The Level II challenges are infeasible given today's computer technology and knowledge.

Broken challenges

1997: Baisley and Harley break ECCp-79.
1997: Harley et al. break ECC2-79.
1998: Harley et al. break ECCp-89.
1998: Harley et al. break ECC2-89.
1998: Harley et al. (1288 computers) break ECCp-97.
1998: Harley et al. (200 computers) break ECC2K-95.
1999: Harley et al. (740 computers) break ECC2-97.
2000: Harley et al. (9500 computers) break ECC2K-108.
2002: Monico et al. (10000 computers) break ECCp-109.
2004: Monico et al. (2600 computers) break ECC2-109.

Updated 2003 document `cert_ecc_challenge.pdf` still said "109-bit Level I challenges are feasible using a very large network . . . 131-bit Level I challenges are expected to be infeasible" etc.

The Certicom challenges ECC2-X

VAM1 research retreat in Lausanne on SHARCS topics.

Decision to analyze the Certicom challenges ECC2K-130, ECC2-131, ECC2K-163, ECC2-163.
Can we break ECC2K-130?
"Infeasible" sounds tempting.

Direct effects:

- Certicom backpedals. Withdraws "infeasible" statement. Instead says that ECC2K-130 "may be within reach."

The Certicom challenges ECC2-X

VAM1 research retreat in Lausanne on SHARCS topics.

Decision to analyze the Certicom challenges ECC2K-130, ECC2-131, ECC2K-163, ECC2-163.
Can we break ECC2K-130?
“Infeasible” sounds tempting.

Direct effects:

- Certicom backpedals. Withdraws “infeasible” statement. Instead says that ECC2K-130 “may be within reach.”
- ECRYPT has several new research papers, starting with paper at SHARCS “The Certicom challenges ECC2-X.”

The target: ECC2K-130

The Koblitz curve $y^2 + xy = x^3 + 1$ over $\mathbf{F}_{2^{131}} = \mathbf{F}_2[z]/(z^{131} + z^{13} + z^2 + z + 1)$ has 4ℓ points, where ℓ is the prime 680564733841876926932320129493409985129 $\approx 2^{129}$.

Certicom generated two random points on the curve and multiplied them by 4, obtaining the following points P, Q:

```
x(P) = 05 1C99BFA6 F18DE467 C80C23B9 8C7994AA
y(P) = 04 2EA2D112 ECEC71FC F7E000D7 EFC978BD
x(Q) = 06 C997F3E7 F2C66A4A 5D2FDA13 756A37B1
y(Q) = 04 A38D1182 9D32D347 BD0C0F58 4D546E9A
```

The challenge:
Find an integer $k \in \{0, 1, \ldots, \ell - 1\}$ such that $[k]P = Q$.
Worthy target:

The target: ECC2K-130

The Koblitz curve $y^2 + xy = x^3 + 1$ over $\mathbf{F}_{2^{131}} = \mathbf{F}_2[z]/(z^{131} + z^{13} + z^2 + z + 1)$ has 4ℓ points, where ℓ is the prime 680564733841876926932320129493409985129 $\approx 2^{129}$.

Certicom generated two random points on the curve and multiplied them by 4, obtaining the following points P, Q:

```
x(P) = 05 1C99BFA6 F18DE467 C80C23B9 8C7994AA
y(P) = 04 2EA2D112 ECEC71FC F7E000D7 EFC978BD
x(Q) = 06 C997F3E7 F2C66A4A 5D2FDA13 756A37B1
y(Q) = 04 A38D1182 9D32D347 BD0C0F58 4D546E9A
```

The challenge:
Find an integer $k \in \{0, 1, \ldots, \ell - 1\}$ such that $[k]P = Q$.
Worthy target: \$ 20000 (but only CAD)

The target: ECC2K-130

The Koblitz curve $y^2 + xy = x^3 + 1$ over $\mathbf{F}_{2^{131}} = \mathbf{F}_2[z]/(z^{131} + z^{13} + z^2 + z + 1)$ has 4ℓ points, where ℓ is the prime $680564733841876926932320129493409985129 \approx 2^{129}$.

Certicom generated two random points on the curve and multiplied them by 4, obtaining the following points P, Q:

```
x(P) = 05 1C99BFA6 F18DE467 C80C23B9 8C7994AA
y(P) = 04 2EA2D112 ECEC71FC F7E000D7 EFC978BD
x(Q) = 06 C997F3E7 F2C66A4A 5D2FDA13 756A37B1
y(Q) = 04 A38D1182 9D32D347 BD0C0F58 4D546E9A
```

The challenge:
Find an integer $k \in \{0, 1, \ldots, \ell - 1\}$ such that $[k]P = Q$.
Worthy target: 128-bit curves have been proposed for real (RFID, TinyTate).

Arithmetic on ECC2K-130

Elements of the Koblitz curve: a special point P_∞, and each $(x_1, y_1) \in \mathbf{F}_{2^{131}} \times \mathbf{F}_{2^{131}}$ satisfying $y_1^2 + x_1 y_1 = x_1^3 + 1$.

Arithmetic on ECC2K-130

Elements of the Koblitz curve: a special point P_∞, and each $(x_1, y_1) \in \mathbf{F}_{2^{131}} \times \mathbf{F}_{2^{131}}$ satisfying $y_1^2 + x_1 y_1 = x_1^3 + 1$.

How to add P_1, P_2:

- $P_1 + P_\infty = P_\infty + P_1 = P_1$; $(x_1, y_1) + (x_1, y_1 + x_1) = P_\infty$.
- If $x_1 \neq 0$ the double $[2](x_1, y_1) = (x_3, y_3)$ is given by

$$x_3 = \lambda^2 + \lambda,\ \ y_3 = \lambda(x_1 + x_3) + y_1 + x_3,\ \text{ where } \lambda = x_1 + \frac{y_1}{x_1}.$$

- If $x_1 \neq x_2$ the sum $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ is given by

$$x_3 = \lambda^2 + \lambda + x_1 + x_2,\ \ y_3 = \lambda(x_1 + x_3) + y_1 + x_3,\ \text{ where } \lambda = \frac{y_1 + y_2}{x_1 + x_2}.$$

  Cost: 1**I** (inversion), 2**M** (multiplications), 1**S** (squaring).
- For an overview of how to perform these operations in other coordinate systems see the EFD:

  `http://hyperelliptic.org/EFD/`

Koblitz curves – the Frobenius endomorphism

- In 1991 Koblitz pointed out that scalar multiplications $[m]P$ can be computed faster on curves

$$E_a : y^2 + xy = x^3 + ax^2 + 1,$$

  where a is restricted to $\{0, 1\}$.
- The main observation is that if $(x_1, y_1) \in E_a(\mathbf{F}_{2^n})$ then also the point $\sigma(P) = (x_1^2, y_1^2)$ is in $E_a(\mathbf{F}_{2^n})$ and these points are related by

$$\sigma^2(P) + [\mu]\sigma(P) + [2]P = P_\infty,$$

  where $\mu = 1$ for $a = 0$ and $\mu = -1$ for $a = 1$. The map σ extends the Frobenius automorphism of $\mathbf{F}_{2^n}$ to $E_a(\mathbf{F}_{2^n})$ and is thus called the Frobenius endomorphism of E_a.

The most important ECDL algorithms

No known index-calculus attack applies to ECC2K-130.
But can still use generic attacks that work in any group:

- The Pohlig–Hellman attack reduces the hardness of the ECDLP to the hardness of the ECDLP in the largest subgroup of prime order: in this case order ℓ.
- The Baby-Step Giant-Step attack finds the logarithm in $\sqrt{\ell}$ steps and $\sqrt{\ell}$ storage by comparing $Q - [jt]P$ (the giant steps) to a sorted list of all $[i]P$ (the baby steps), where $0 \leq i, j \leq \lceil\sqrt{\ell}\rceil$ and $t = \lceil\sqrt{\ell}\rceil$.
- Pollard's rho and kangaroo methods also use $O(\sqrt{\ell})$ steps but require constant memory—much less expensive! The kangaroo method would be faster if the logarithm were known to lie in a short interval; for us rho is best.
- Multiple-target attacks: not relevant here.

Pollard's rho method

Make a pseudo-random walk in $\langle P \rangle$, where the next step depends on current point: $P_{i+1} = f(P_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk has now entered a cycle.
Cycle-finding algorithm (e.g., Floyd) quickly detects this.

Pollard's rho method

Make a pseudo-random walk in $\langle P \rangle$, where the next step depends on current point: $P_{i+1} = f(P_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk has now entered a cycle.
Cycle-finding algorithm (e.g., Floyd) quickly detects this.

Assume that for each point we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $P_i = [a_i]P + [b_i]Q$. Then $P_i = P_j$ means that

$$[a_i]P + [b_i]Q = [a_j]P + [b_j]Q \quad \text{so} \quad [b_i - b_j]Q = [a_j - a_i]P.$$

If $b_i \neq b_j$ the ECDLP is solved: $k = (a_j - a_i)/(b_i - b_j)$.

Pollard's rho method

Make a pseudo-random walk in $\langle P\rangle$, where the next step depends on current point: $P_{i+1} = f(P_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk has now entered a cycle.
Cycle-finding algorithm (e.g., Floyd) quickly detects this.

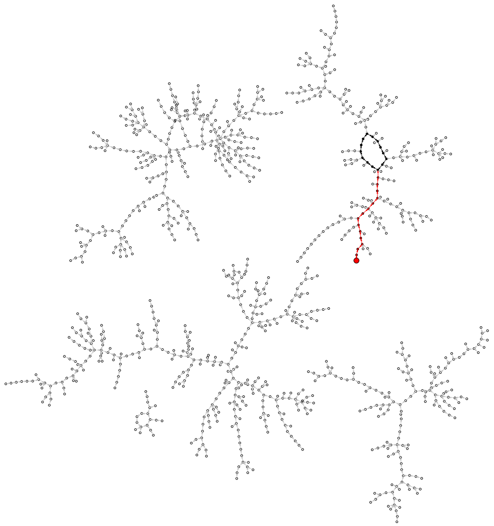
Assume that for each point we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $P_i = [a_i]P + [b_i]Q$. Then $P_i = P_j$ means that

$$[a_i]P + [b_i]Q = [a_j]P + [b_j]Q \quad \text{so} \quad [b_i - b_j]Q = [a_j - a_i]P.$$

If $b_i \neq b_j$ the ECDLP is solved: $k = (a_j - a_i)/(b_i - b_j)$.

e.g. "Adding walk": Start with $P_0 = P$ and put $f(P_i) = P_i + [c_r]P + [d_r]Q$ where $r = h(P_i)$.

A rho within a random walk on 1024 elements



Method is called rho method because of the shape.

Parallel collision search

Running Pollard's rho method on N computers gives speedup of $\approx \sqrt{N}$ from increased likelihood of finding collision.

Want better way to spread computation across clients.
Want to find collisions between walks on different machines, without frequent synchronization!

Parallel collision search

Running Pollard's rho method on N computers gives speedup of $\approx \sqrt{N}$ from increased likelihood of finding collision.

Want better way to spread computation across clients. Want to find collisions between walks on different machines, without frequent synchronization!

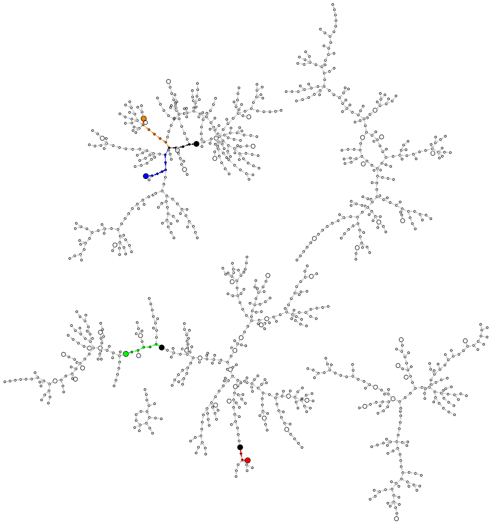
Perform walks with different starting points but same update function on all computers. If same point is found on two different computers also the following steps will be the same.

Terminate each walk once it hits a distinguished point. Attacker chooses definition of distinguished points; can be more or less frequent. Do not wait for cycle.

Collect all distinguished points in central database.

Expect collision within $O(\sqrt{\ell}/N)$ iterations. Speedup $\approx N$.

Short walks ending in distinguished points



Blue and orange paths found the same distinguished point!

Equivalence classes

P and $-P$ have same x-coordinate. Search for x-coordinate collision. Search space for collisions is only $\ell/2$; this gives factor $\sqrt{2}$ speedup ... provided that $f(P_i) = f(-P_i)$.

Solution: $f(P_i) = |P_i| + [c_r]P + [d_r]Q$ where $r = h(|P_i|)$. Define $|P_i|$ as, e.g., lexicographic minimum of $P_i, -P_i$.

Equivalence classes

P and $-P$ have same x-coordinate. Search for x-coordinate collision. Search space for collisions is only $\ell/2$; this gives factor $\sqrt{2}$ speedup … provided that $f(P_i) = f(-P_i)$.

Solution: $f(P_i) = |P_i| + [c_r]P + [d_r]Q$ where $r = h(|P_i|)$. Define $|P_i|$ as, e.g., lexicographic minimum of $P_i, -P_i$.

Problem: this walk can run into fruitless cycles!
If there are S different steps $[c_r]P + [d_r]Q$ then with probability $1/(2S)$ the following happens for some step:

$$\begin{aligned} P_{i+2} &= P_{i+1} + [c_r]P + [d_r]Q \\ &= -(P_i + [c_r]P + [d_r]Q) + [c_r]P + [d_r]Q = -P_i, \end{aligned}$$

i.e. $|P_i| = |P_{i+2}|$. Get $|P_{i+3}| = |P_{i+1}|$, $|P_{i+4}| = |P_i|$, etc.
Can detect and fix, but requires attention.

Equivalence classes for Koblitz curves

More savings: P and $\sigma^i(P)$ have $x(\sigma^j(P)) = x(P)^{2^j}$.

Reduce number of iterations by another factor $\sqrt{n}$ by considering equivalence classes under Frobenius and $\pm$.

Need to ensure that the iteration function satisfies $f(P_i) = f(\pm\sigma^j(P_i))$ for any j.

Equivalence classes for Koblitz curves

More savings: P and $\sigma^i(P)$ have $x(\sigma^j(P)) = x(P)^{2^j}$.

Reduce number of iterations by another factor $\sqrt{n}$ by considering equivalence classes under Frobenius and $\pm$.

Need to ensure that the iteration function satisfies $f(P_i) = f(\pm\sigma^j(P_i))$ for any j.

Could again define adding walk starting from $|P_i|$.
Redefine $|P_i|$ as canonical representative of class containing P_i: e.g., lexicographic minimum of P_i, $-P_i$, $\sigma(P_i)$, etc.

Iterations now involve many squarings,
but squarings are not so expensive in characteristic 2.

Our choice of iteration function

In normal basis, $x(P)$ and $x(P)^{2^j}$ have same Hamming weight $\mathrm{HW}(x(P))$. Convenient to use this to determine iteration.

Our iteration function—note that $\mathrm{HW}(x(P))$ is always even:

$$P_{i+1} = P_i + \sigma^j(P_i),$$

where $j = (\mathrm{HW}(x(P))/2 \bmod 8) + 3$.

Iteration consists of

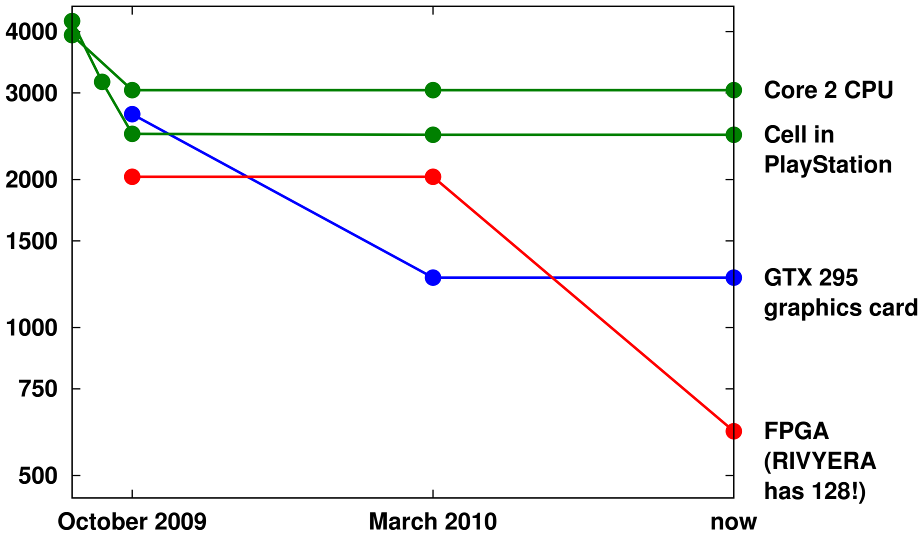
- computing the Hamming weight $\mathrm{HW}(x(P))$ of the normal-basis representation of $x(P)$;
- checking for distinguished points (is $\mathrm{HW}(x(P)) \leq 34$?);
- computing j and $P + \sigma^j(P)$.

Choice of j avoids short, fruitless cycles.

Some highlights

- Detailed analysis of randomness of iteration function.
- Could increase randomness of the walk but then iteration function gets slower. Optimized:
  $$\text{time per iteration} \times \#\ \text{iterations}$$
- Do not remember multiset of j's; instead recompute this from seed when collision is found (cheaper, less storage).
- Comparative study of normal basis and polynomial basis representation; new: optimal polynomial bases.
- For Cell processor (chip in PlayStation 3) fierce battle between bitsliced and non-bitsliced implementation. Result: much faster implementation! (Bitsliced won.)
- Assembly language for GPUs and qhasm version. Get control over powerful beast.
- FPGA implementation of Shokrollahi multiplier: big speed-up, useful also for constructive ECC.

Faster implementations



Number of cards or chips needed for $68 \cdot 10^9$ iterations/second.

Overall speedup

E.g. 2466 Cell CPUs for a year: i.e., 900000 machine days.
Recall Certicom's estimate: 2700000000 machine days.

Overall speedup

E.g. 2466 Cell CPUs for a year: i.e., 900000 machine days.
Recall Certicom's estimate: 2700000000 machine days.

"That's unfair! Computers ten years ago were very slow!"

Overall speedup

E.g. 2466 Cell CPUs for a year: i.e., 900000 machine days.
Recall Certicom's estimate: 2700000000 machine days.

"That's unfair! Computers ten years ago were very slow!"
Indeed, Certicom's "machine" was a 100MHz Pentium.
Today's Cell has several cores, each running at 3.2GHz.
Scale by counting cycles . . . and we're still $15\times$ faster.

Overall speedup

E.g. 2466 Cell CPUs for a year: i.e., 900000 machine days.
Recall Certicom's estimate: 2700000000 machine days.

"That's unfair! Computers ten years ago were very slow!"
Indeed, Certicom's "machine" was a 100MHz Pentium.
Today's Cell has several cores, each running at 3.2GHz.
Scale by counting cycles . . . and we're still $15\times$ faster.

"Computers ten years ago didn't do much work per cycle!"

Overall speedup

E.g. 2466 Cell CPUs for a year: i.e., 900000 machine days.
Recall Certicom's estimate: 2700000000 machine days.

"That's unfair! Computers ten years ago were very slow!"
Indeed, Certicom's "machine" was a 100MHz Pentium.
Today's Cell has several cores, each running at 3.2GHz.
Scale by counting cycles . . . and we're still 15$\times$ faster.

"Computers ten years ago didn't do much work per cycle!"
True for the Pentium . . . but not for Harley's Alpha,
which had 64-bit registers, 4 instructions/cycle, etc.

Harley's ECC2K-108 software uses 1651 Alpha cycles/iteration.
We ran the same software on a Core 2: 1800 cycles/iteration.
We also wrote our own polynomial-basis ECC2K-108 software:
on the same Core 2, fewer than 500 cycles/iteration.

Running the attack

Is ECC2K-130 feasible for a serious attacker? Obviously.

Running the attack

Is ECC2K-130 feasible for a serious attacker? Obviously.
Is ECC2K-130 feasible for a big public Internet project? Yes.

Running the attack

Is ECC2K-130 feasible for a serious attacker? Obviously.
Is ECC2K-130 feasible for a big public Internet project? Yes.
Is ECC2K-130 feasible for us? We think so.
To prove it we're running the attack.

Running the attack

Is ECC2K-130 feasible for a serious attacker? Obviously.
Is ECC2K-130 feasible for a big public Internet project? Yes.
Is ECC2K-130 feasible for us? We think so.
To prove it we're running the attack.

Eight central servers receive points, pre-sort the points into 8192 RAM buffers, flush the buffers to 8192 disk files.

Periodically read each file into RAM, sort, find collisions.
Also double-check random samples for validity.

Several sites contribute points, including several clusters. E.g. test runs on first generation of PRACE clusters

`http://www.prace-project.eu`

Each packet is encrypted, authenticated, verified, decrypted using `http://nacl.cace-project.eu`; costs 16 bytes.
Total block cost: 1090-byte IP packet plus 66-byte ack.

Reports so far (2010.09.07 tallies from servers)

- from site "c": 21820833792 bytes; GPUs, Core 2
- from site "L": 17830614016 bytes; Opteron
- from site "ℓ": 14669790208 bytes; Cell (PS3), Core 2
- from site "d": 3939812352 bytes
- from site "e": 3839403008 bytes; GPUs, Core 2
- from site "j": 3605491712 bytes; Cell blade
- from site "t": 3505213440 bytes; Core 2, Phenom II, GPUs
- from site "G": 3476676608 bytes
- from site "p": 3038750720 bytes
- from site "z": 1732281344 bytes
- from site "B": 1604201472 bytes
- from site "b": 507629568 bytes
- from site "a": 117901312 bytes
- from site "n": 86499328 bytes

Top priority: Get RIVYERA (128 FPGAs) running!

Get more details, and watch our progress!

`http://ecc-challenge.info`
`https://twitter.com/ECCchallenge`

Papers and preprints:

- "The Certicom challenges ECC2-X" (SHARCS 2009)
- "ECC2K-130 on Cell CPUs" (AFRICACRYPT 2010)
- "Type-II optimal polynomial bases" (WAIFI 2010)
- "Breaking elliptic curve cryptosystems using reconfigurable hardware" (FPL 2010)
- "ECC2K-130 on NVIDIA GPUs"
- "Usable assembly language for GPUs"
- The whole attack in progress: "Breaking ECC2K-130"