Accelerating Pollard's Log Methods for ABC's

Robert Gallant
Robert Lambert
Scott Vanstone

Certicom Research

{rgallant,rlambert,savansto}@certicom.com

Sept. 15, 1998

Abstract

The structure of Anomalous Binary Curves (ABC's) can be used to accelerate Pollard's discrete logarithm methods on these curves. Equivalence classes on the points are defined, and a well-defined iteration between classes is developed. A $\sqrt{2m}$ speedup for an ABC over $\mathbb{F}_{2^m}$ is obtained.

Elliptic Curve Discrete Logarithm

$$Q \in < P >$$

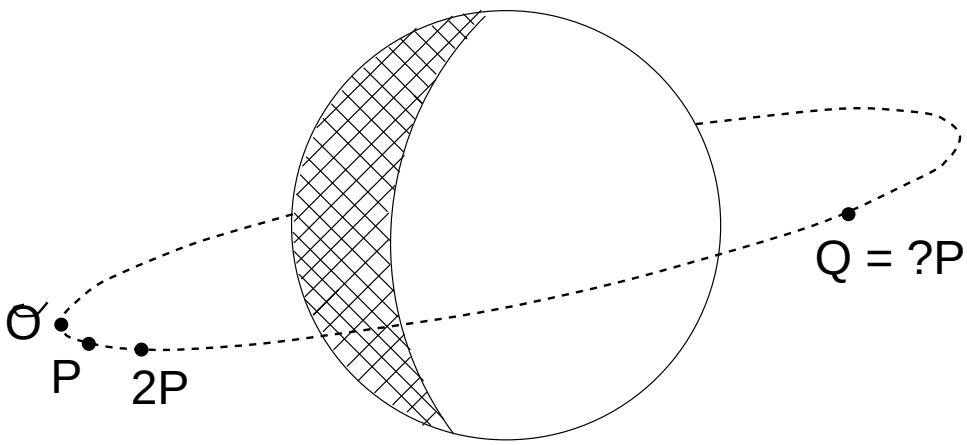

Figure 1: Finding Q in $< P >$

Pollard's Methods

$$Q \in <P>,\ Q = ?P$$

Position: $T = [u, v, R = uP + vQ]$

$$T_{i+1} \leftarrow \begin{cases} T_i + [0, 1, Q]; & R(T_i) \in S_0 \\ T_i + T_i; & R(T_i) \in S_1 \\ T_i + [1, 0, P]; & R(T_i) \in S_2 \end{cases},$$

where S_0, S_1, S_2 partition $<P>$.

Assuming the iteration behaves randomly:

- *Rho* in expected $\sqrt{\frac{\pi\ |<P>|}{2}}$ steps.
- *Cycle* found with Floyd's cycle finding method.

Pollard's Methods (cont'd)

More generally:

$$T_{i+1} \leftarrow T_i + C_{\mathcal{H}(\mathcal{L}(T_i))},$$

where:

C_j are fixed combinations of P, Q, T,

$\mathcal{H}$ is an (ordinary) hash function,

and $\mathcal{L}$ is a label (representation) for the element.

Solving for Logarithms

Once a collision is found, $R_i = R_j$, then

$$\mu_i^P P + \mu_i^Q Q = \mu_j^P P + \mu_j^Q Q$$

$$(\mu_i^P - \mu_j^P) P = -(\mu_i^Q - \mu_j^Q) Q$$

$$\Delta^P P = -\Delta^Q Q$$

$$\log_P(Q) = -(\Delta^P / \Delta^Q) \pmod{n}, \quad \text{if } \Delta^Q \neq 0.$$

A Single Machine

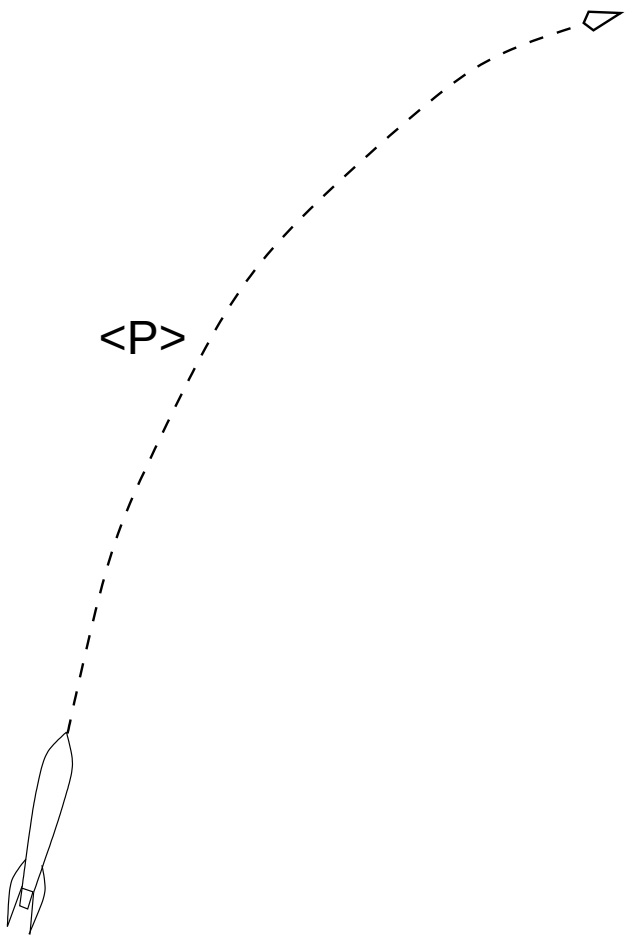

Figure 2: Trajectory in $<P>$.

Pollard Parallelized

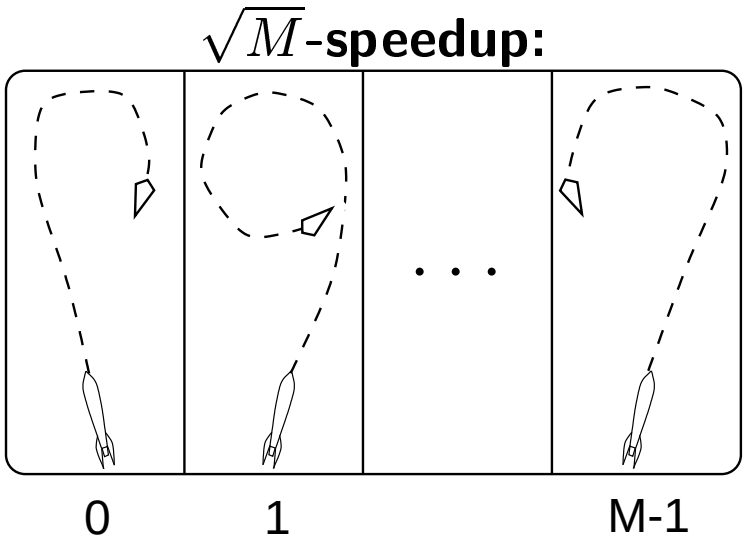

Figure 3: M machines operating independently.

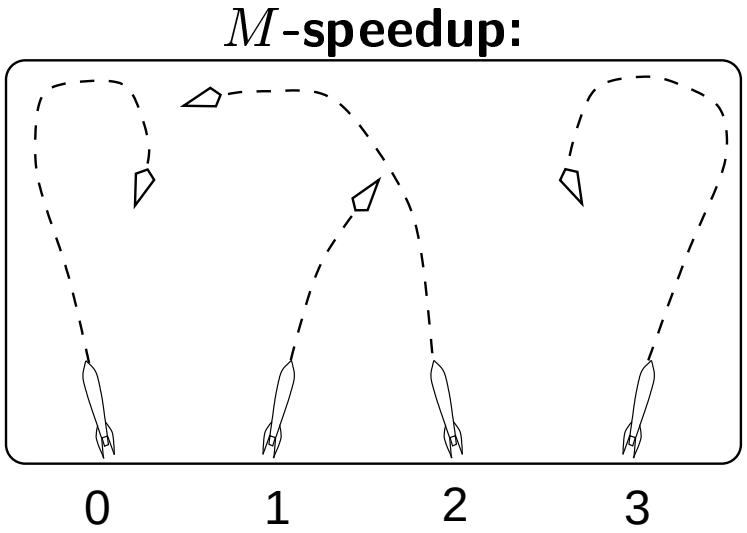

Figure 4: M machines with global information.

Efficient Global Information

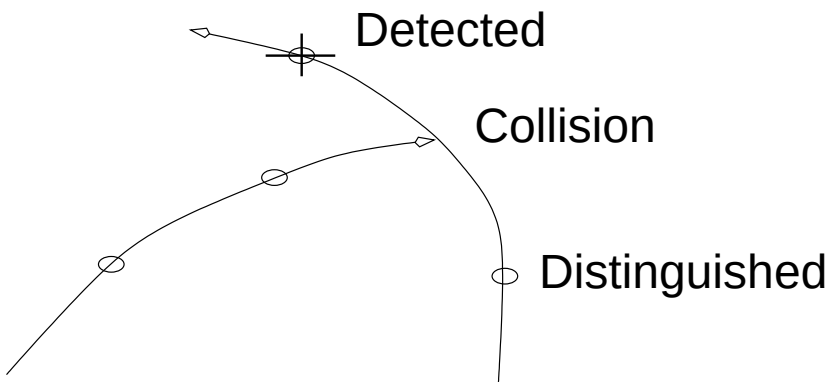

Figure 5: Distinguished points.

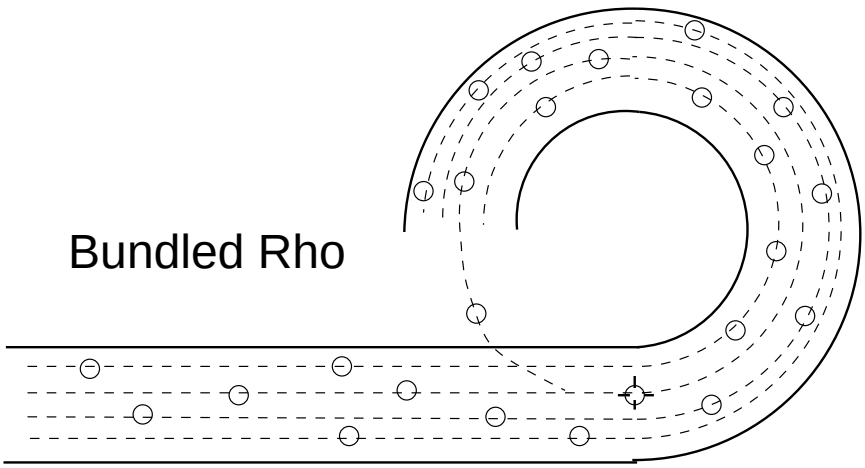

Figure 6: Bundled Rho.

Can we do better?

Shoup [96] shows that if we have a *black-box* group then the best algorithm is $O(\sqrt{p})$, p prime order of the group. To do better, we need to use the structure of the group.

Log-Useful Structure of ABC's

Anomalous binary curves (ABC's)
have a log-useful structure:

$$Q = (x, y)$$

$$-Q = (x, x + y)$$

$$\lambda Q = (x^2, y^2)$$

$$\lambda^2 Q = (x^{2^2}, y^{2^2})$$

This is a relation between the logarithm
and the structure of the representation:

$$\lambda^r Q = (x^{2^r}, y^{2^r})$$

In a normal basis:

$\lambda^r Q = (\overset{r}{\hookleftarrow} x, \overset{r}{\hookleftarrow} y)$, where $\overset{r}{\hookleftarrow}$ is an r-rotation

Equivalence Classes

Assuming $\lambda^k \neq -1$, $\mathbb{F}_{2^m}$:

$$[\mathcal{O}]: \quad \{ \quad \mathcal{O} \quad \}$$

$$[P_0]: \quad \{ \quad P_0, \quad \lambda P_0, \quad \lambda^2 P_0, \quad \cdots, \quad \lambda^{m-1} P_0,$$
$$-P_0, \quad -\lambda P_0, \quad -\lambda^2 P_0, \quad \cdots, \quad -\lambda^{m-1} P_0 \quad \}$$

$$[P_1]: \quad \{ \quad P_1, \quad \lambda P_1, \quad \lambda^2 P_1, \quad \cdots, \quad \lambda^{m-1} P_1,$$
$$-P_1, \quad -\lambda P_1, \quad -\lambda^2 P_1, \quad \cdots, \quad -\lambda^{m-1} P_1 \quad \}$$

$\vdots$

equivalence classes.

In a normal-basis representation, the x-coords of $[P_i]$ are rotations of each other.

If we can randomly iterate over the classes, instead of the points, then the number of elements are reduced by $2m$. This accelerates Pollard's method by $\sqrt{2m}$.

Equivalence Classes (pictorial)

$\sqrt{2m}$-speedup:

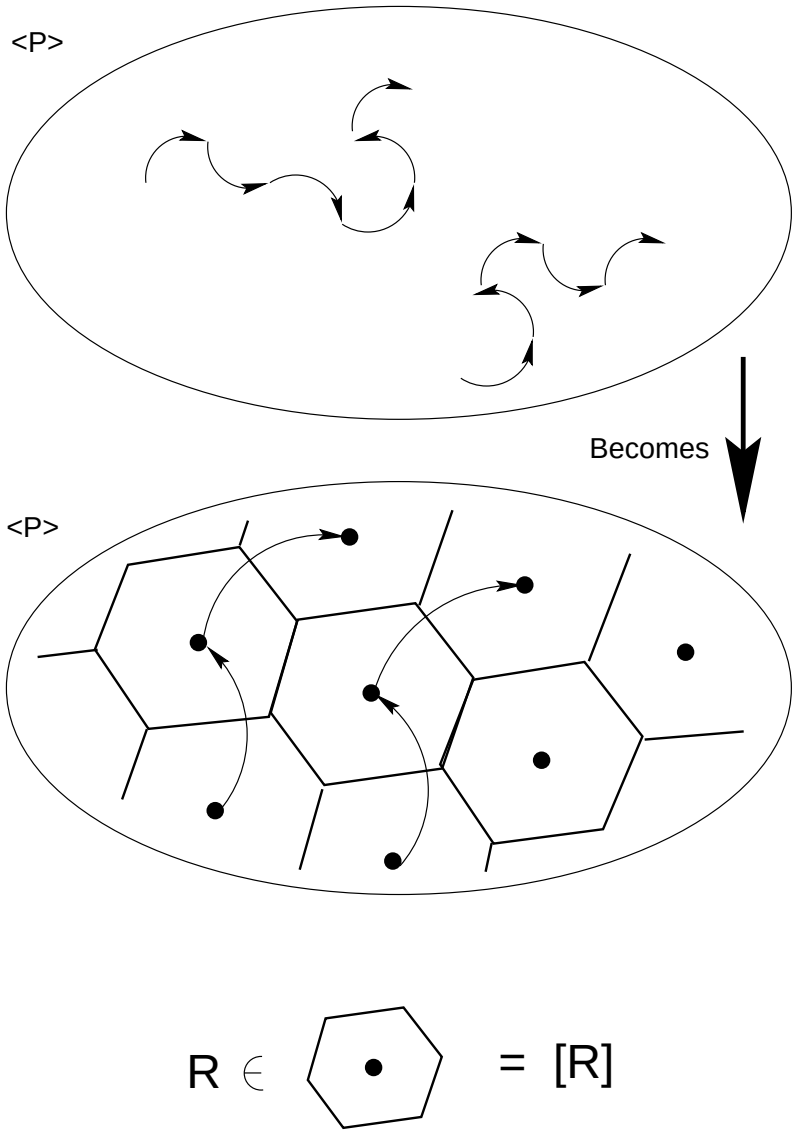

Figure 7: Equivalence classes in $<P>$

Well-defined Iterations

Well-defined iteration over Equivalence Classes

Lemma 1. *The iteration:* $f : R_{i+1} \leftarrow \mu_{[R_i]} \cdot R_i$, *where* $\mu_{[R_i]}$ *is a multiple depending on* $[R_i]$, *though defined on elements* R, *is well defined on equivalence classes* $[R]$.

Proof. if $R' \in [R]$ then $R' = \pm 1 \lambda^r R$.

$$f(R) = \mu_{[R]} R$$

$$f(R') = \mu_{[R']} R' = \mu_{[R]} (\pm 1) \lambda^r R \in [\mu_{[R]} R]$$

Therefore

$$[f(R)] = [f(R')]. \quad \square$$

Building $\mu_{[R]}$

A convenient way to build $\mu_{[R]}$:

if $\mathcal{L}[R]$ is a label for $[R]$,

(e.g. lexicographically least x^{2^i}, or least rotation if x in a normal basis)

then a suitable multiple is

$$\mu_{[R]} = (1 + \lambda^{\mathcal{H}(\mathcal{L}[R])}),$$

where $\mathcal{H}$ is an (ordinary) hash function taking $\mathcal{L}$ to $[0, m-1]$.

The function

$$f : R_{i+1} \leftarrow R_i + \lambda^{\mathcal{H}(\mathcal{L}[R])} R_i$$

is efficiently computed.

Why well defined?

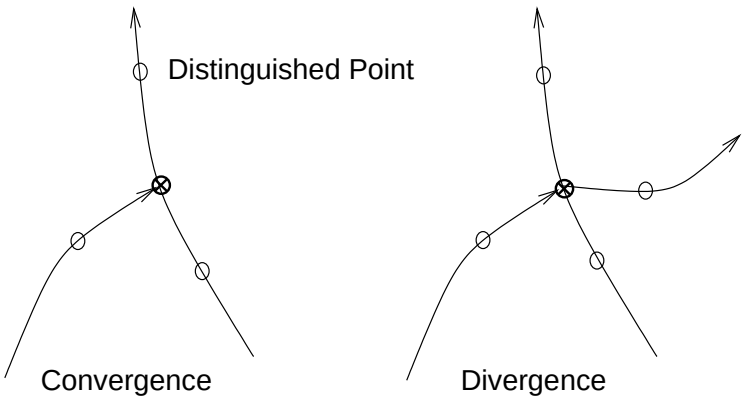

Figure 8:

Alternative Method:

Let $\overline{R}$ be a canonical representative for $[R]$, e.g. least lexicographical (x, y)-pair in $[R]$.

Pollard-like iteration:

$T_{i+1} \leftarrow \overline{T_i + \mu^P(T_i)P + \mu^Q(T_i)Q}$, where μ^P and μ^Q are multiples depending on T_i, is a well-defined iteration.

This method independently discovered by Wiener and Zuccherato. This method suffers from short, unproductive cycles which require corrective action.

Useless Cycles

If T_i (canonical), $T_i \in S_j$, then

$$T_{i+1} \leftarrow \overline{T_i + C_j}.$$

If the normalization is -1, then

$$T_{i+1} = -T_i - C_j.$$

If $T_{i+1} \in S_j$, then

$$\begin{aligned} T_{i+2} &\leftarrow \overline{T_{i+1} + C_j} \\ &= \overline{-T_i - C_j + C_j} \\ &= \overline{-T_i} \\ &= T_i, \end{aligned}$$

a useless cycle.

Implementation

Data:

i-th machine begins at random u^i, v^i:

$$R_0^i \leftarrow u^i P + v^i Q$$

$$\begin{aligned} R_j^i &= (1+\lambda^0)^{e_0}(1+\lambda^1)^{e_1}\cdots(1+\lambda^{m-1})^{e_{m-1}} R_0^i \\ &= \mu_j^i R_0^i \end{aligned}$$

$$e = [e_0, e_1, e_2, \cdots e_{m-1}]$$

Central Machine:

List of initial conditions, plus distinguished points:

Machine i:

$[[i, u^i, v^i]; [\mathcal{L}[R_{d_0}^i], \mu_{d_0}^i], [\mathcal{L}[R_{d_1}^i], \mu_{d_1}^i], \cdots]$

Searching

The i-th machine:

1. calculates $R^i_{j+1} \leftarrow (1+\lambda^r)R^i_j$

2. updates e

3. if R^i_{j+1} is a distinguished point $R^i_{d_k}$:
   (a) calculates $\mu^i_{d_k}$ from e.
   (b) sends $[i, \mathcal{L}[R^i_{d_k}], \mu^i_{d_k}]$ to the central machine.

For efficiency, the vector e can be updated at each iteration, and the multiple $\mu^i_{d_k}$ calculated only on distinguished points. Also, e can be reset at each distinguished point, and the new multiple calculated from the previous multiple at a distinguished point. Small tables of some powers of $(1+\lambda^r)$ are useful for this calculation.

Note, less storage is required on the central machine than for Pollard-like iterations.

Solving

On a collision

$$\mathcal{L}[R^i] = \mathcal{L}[R^j].$$

After a short search, s, t can be found such that

$$\mu^i(u^iP + v^iQ) = (-1)^s(\lambda)^t\mu^j(u^jP + v^jQ)$$

$$\frac{\mu^i u^i P - (-1)^s(\lambda)^t\mu^j u^j}{(-1)^s(\lambda)^t\mu^j v^j - \mu^i v^i}P = Q,$$

if $(-1)^s(\lambda)^t\mu^j v^j - \mu^i v^i \neq 0$.

Then

$$\log_P(Q) = \frac{\mu^i u^i P - (-1)^s(\lambda)^t\mu^j u^j}{(-1)^s(\lambda)^t\mu^j v^j - \mu^i v^i} \pmod{n}.$$

Conclusions

The structure of ABC's can be used to accelerate the parallel Pollard method by $\sqrt{2m}$.

For example,

- Certicom **ECC2K-95** challenge from 2^{48} to 2^{44} steps,
- Certicom **ECC2K-108** challenge from 2^{54} to 2^{50} steps.

Indeed, the **ECC2K-95** was solved in roughly 2^{44} by R. Harley of INRIA and his collaborators (including British Telecom), who developed similar ideas independently.

Open Questions

Can other equivalence classes be used to speed logarithms?

Example: $\{2^i Q\}$. No advantage.