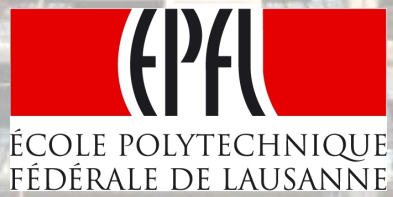
How to solve a 112-bit ECDLP using game consoles

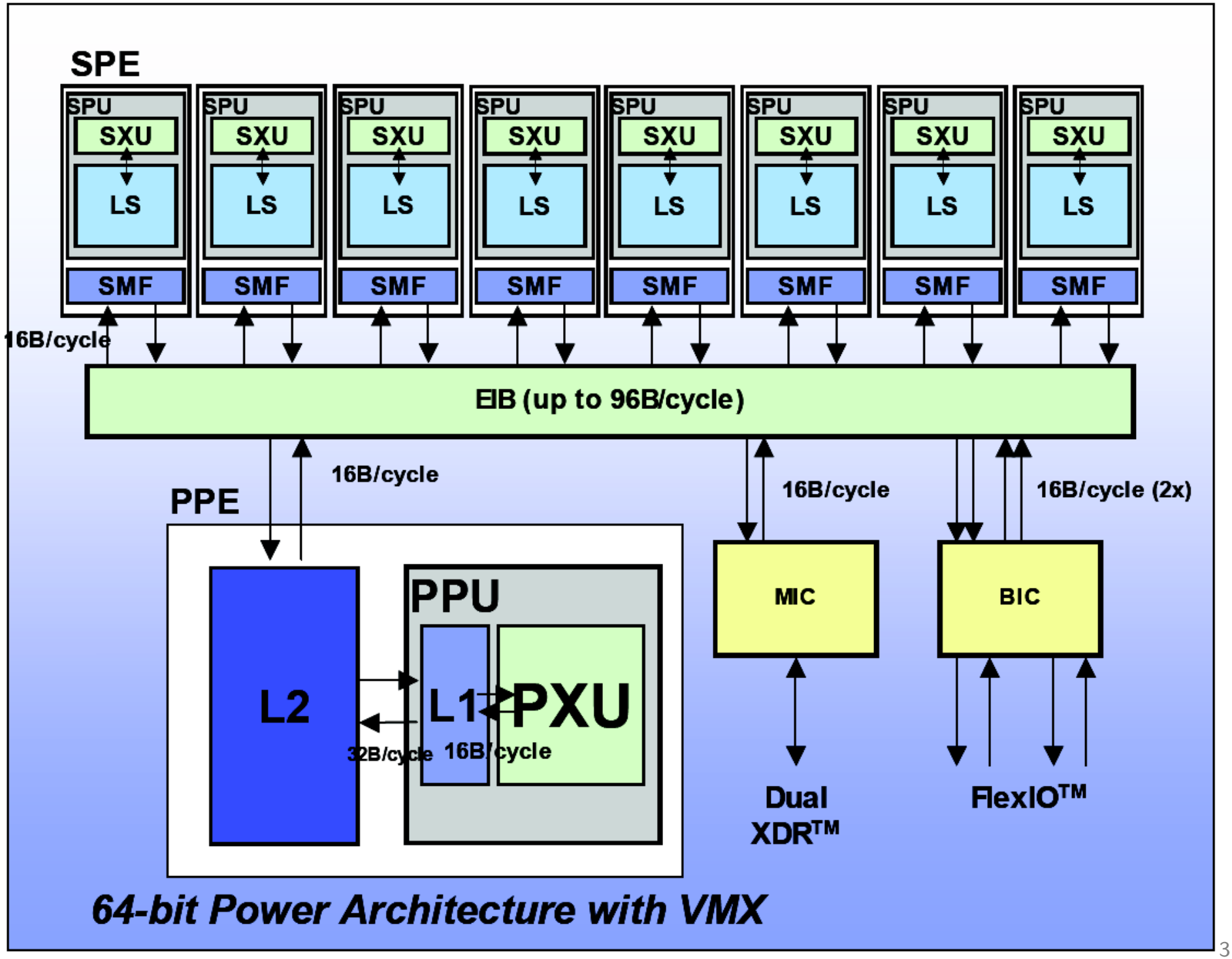
Joppe W. Bos

Laboratory for Cryptologic Algorithms
EPFL, Station 14, CH-1015 Lausanne, Switzerland



Outline

- The Cell Broadband Engine Architecture
  - H. P. Hofstee. *Power efficient processor architecture and the Cell processor*. HPCA 2005, pages 258–262, 2005.
- Project 1: 112-bit prime field ECDLP
- Project 2: On the Use of the Negation Map in the Pollard Rho Method


SPE
SPU
SXU
LS
SMF
16B/cycle
EIB (up to 96B/cycle)
16B/cycle
16B/cycle
16B/cycle (2x)
PPE
L2
PPU
L1
PXU
32B/cycle
16B/cycle
MIC
BIC
Dual XDR™
FlexIO™
64-bit Power Architecture with VMX

Cell Availability

	PS3 slim	PS3 discontinued	PCIe	BladeServer QS22*
Speed	3.2GHz	3.2GHz	2.8GHz	3.2GHz
#SPEs	6	6	8	16
Memory	$\approx$256MB	$\approx$256MB	4GB	$\leq$32GB
Price	\$299.99	\$100 – \$300	$\approx$ \$8k	\$10k – \$14k
Power	250W	280W	210W	230W
Compatibility	PSOne	PSOne, Linux	Linux	Linux

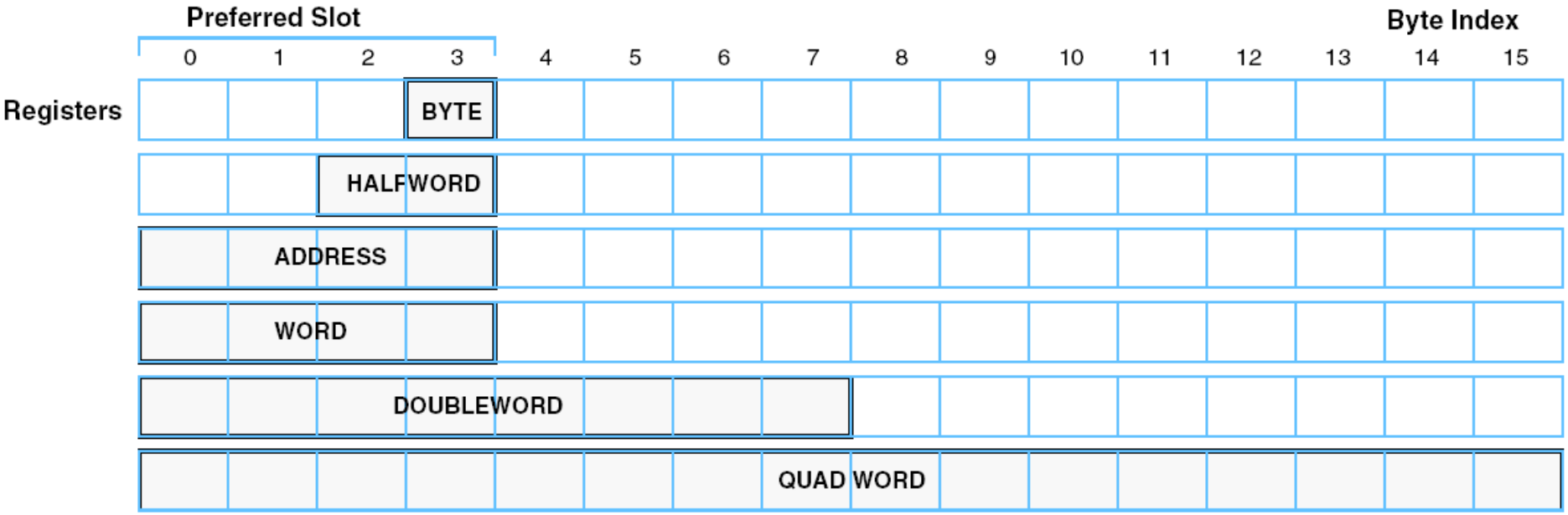
$\star$ IBM PowerXCell 8i processor, offering five times the double precision performance of the previous Cell/B.E. processor.

Cell architecture, the SPEs

The SPEs contain

- a Synergistic Processing Unit (SPU)
  - Access to 128 registers of 128-bit
  - SIMD operations
  - Dual pipeline (odd and even)
  - In-order processor
- 256 KB of fast local memory (Local Store)
- Memory Flow Controller (MFC)
  - Direct Memory Access (DMA) controller
  - Handles synchronization operations to the other SPUs and the PPU
  - DMA transfers are independent of the SPU program execution

SPU registers



- Byte (8-bit): 16-way SIMD
- Half-word (16-bit): 8-way SIMD
- Word (32-bit): 4-way SIMD

Special SPU instructions

All distinct binary operations $f : \{0,1\}^2 \to \{0,1\}$ are present.
Furthermore:

shuffle bytes	add/sub extended
or across	count leading zeros
average of two vectors	count ones in bytes
select bits	gather lsb
carry/borrow generate	sum bytes
multiply and add	multiply and subtract

only $16 \times 16 \to 32$-bit multiplication
but, $16 \times 16 + 32 \to 32$-bit multiply-and-add instruction

Special SPU instructions

All distinct binary operations $f : \{0,1\}^2 \to \{0,1\}$ are present.
Furthermore:

shuffle bytes	add/sub extended
or across	count leading zeros
average of two vectors	count ones in bytes
select bits	gather lsb
carry/borrow generate	sum bytes
multiply and add	multiply and subtract

only 4-way SIMD $16 \times 16 \to 32$-bit multiplication
but, 4-way SIMD $16 \times 16 + 32 \to 32$-bit multiply-and-add instruction

Considerations

- Branching
  - No “smart” dynamic branch prediction
  - Instead “prepare-to-branch” instructions to redirect instruction prefetch to branch targets
- Memory
  - The executable **and** all data should fit in the LS
  - *Or* perform manual DMA requests to the main memory (max. 214 MB)
- Instruction set limitations
  - $16 \times 16 \rightarrow 32$ bit multipliers (4-SIMD)
- Challenge
  - One odd and one even instruction can be dispatched per clock cycle.

LACAL setup

- Physically in the cluster room: 190 PS3s
- 6×4 PS3s in the PlayLaB (attached to the cluster)
- 5 PS3 in our offices for programming purposes
- $\Rightarrow$ 219 PS3s in total.


{ free busy full down disabled absent }
jubos@frontend ~ $ cluster-nodes
{ free busy full down disabled absent }
16 |0123|4567|89AB|CDEF|
15 |0123|4567|89AB|CDEF|
14 |0123|4567|89AB|CDEF|
13 |0123|4567|89AB|CDEF|
12 |0123|4567|89AB|CDEF|
11 |0123|4567|89AB|CDEF|
10 |0123|4567|89AB|CDEF|
9 |0123|4567|89AB|CDEF|
8
7 |0123|4567|89AB|CDEF|
6 |0123|4567|89AB|CDEF|
5 |0123|4567|89AB|CDEF|
4 |0123|4567|89AB|CDEF|
3 |0123|4567|89AB|CDEF|
2 |0123|4567|89AB|CDEF|
1 |0123|4567|89AB|CDEF|
Master: M Slave: S
jubos@frontend ~ $

Outline

- The Cell Broadband Engine Architecture
- Project 1: 112-bit prime field ECDLP
  - Joppe W. Bos, Marcelo E. Kaihara, Thorsten Kleinjung, Arjen K. Lenstra, Peter L. Montgomery: *Solving a 112-bit Prime Elliptic Curve Discrete Logarithm Problem on Game Consoles using Sloppy Reduction* In The International Journal of Applied Cryptography, 2011 (to appear)
- Project 2: On the Use of the Negation Map in the Pollard Rho Method

The Elliptic Curve Discrete Logarithm Problem (ECDLP)

The setting:

- E is an elliptic curve over $\mathbf{F}_p$ with p odd prime.
- $P \in E(\mathbf{F}_p)$ a point of (prime) order n.
- $Q = k \cdot P \in \langle P \rangle$.

Problem: Given E, p, n, P and Q what is k?

ECDLP Parameters

Certicom Challenge

- Solve the ECDLP for EC over $\mathbf{F}_p$ (p odd prime) and $\mathbf{F}_{2^m}$.
- 109-bit prime challenge solved in November 2002 by Chris Monico
  Required time: 4000-5000 PCs working 24/7 for one year.
- Next challenge is an EC over an 131-bit prime field

The 131-bit challenge requires 2000 times the effort of the 109-bit

ECDLP Parameters

ECC Standards

- Standard for Efficient Cryptography (SEC),
  SEC2: Recommended Elliptic Curve Domain Parameters
  Prime fields bit length: { 112, 128, 160, 192, 224, 256, 384, 521 }
- Wireless Transport Layer Security Specification
  Prime fields bit length: { 112, 160, 224 }
- Digital Signature Standard (FIPS PUB 186-3)
  Prime fields bit length: { 192, 224, 256, 384, 521 }

ECDLP Parameters

ECC Standards

- Standard for Efficient Cryptography (SEC),
  SEC2: Recommended Elliptic Curve Domain Parameters
  Prime fields bit length: { 112, 128, 160, 192, 224, 256, 384, 521 }
- Wireless Transport Layer Security Specification
  Prime fields bit length: { 112, 160, 224 }
- Digital Signature Standard (FIPS PUB 186-3)
  Prime fields bit length: { 192, 224, 256, 384, 521 }

How fast can we solve this 112-bit ECDLP?

How fast can we solve an 112-bit ECDLP?

Pollard rho

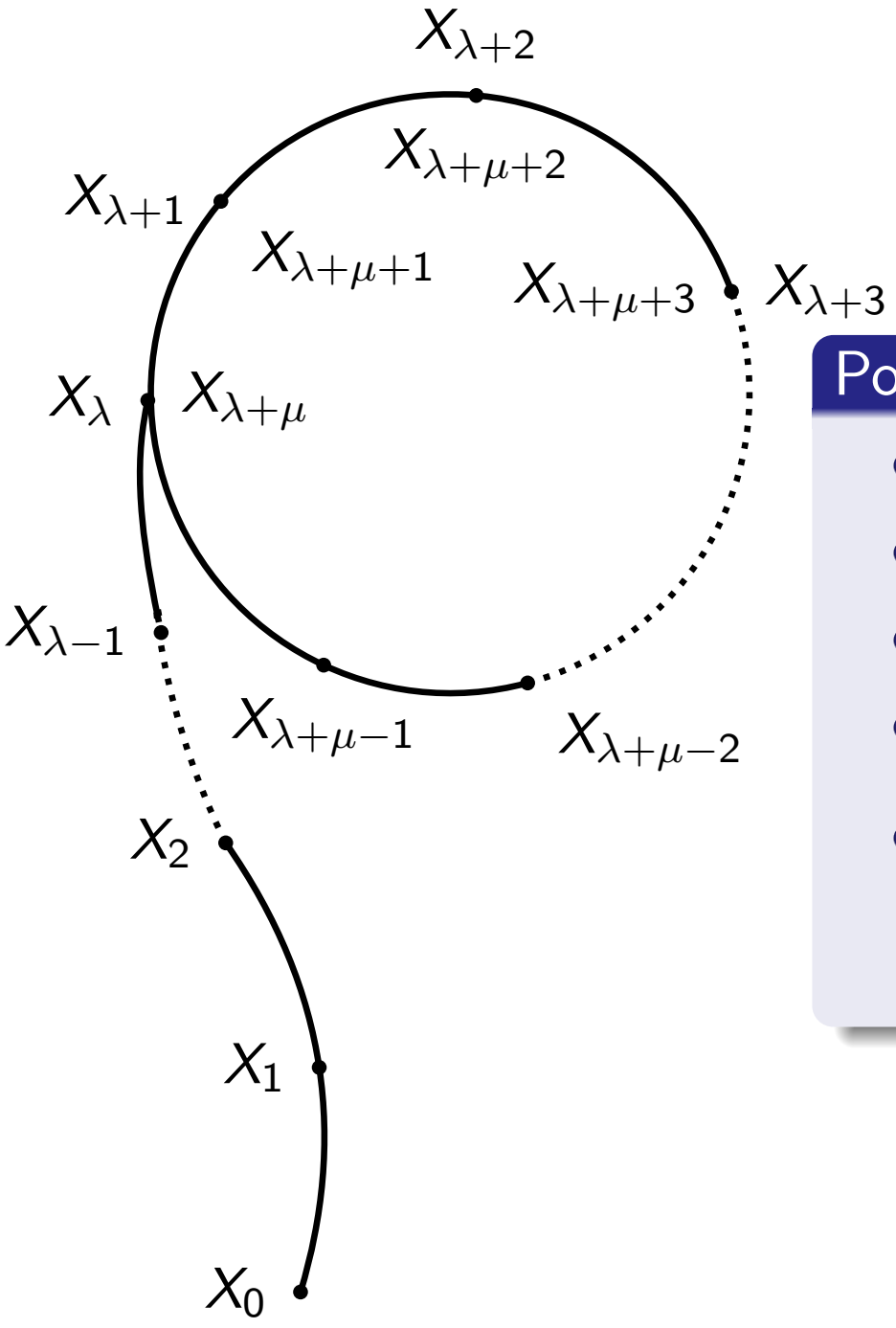
The most efficient algorithm in the literature (for generic curves) is Pollard rho. The underlying idea of this method is to search for two distinct pairs $(c_i, d_i), (c_j, d_j) \in \mathbf{Z}/n\mathbf{Z} \times \mathbf{Z}/n\mathbf{Z}$ such that

$$c_i \cdot P + d_i \cdot Q = c_j \cdot P + d_j \cdot Q$$

$$(c_i - c_j) \cdot P = (d_j - d_i) \cdot Q = (d_j - d_i)k \cdot P$$

$$k \equiv (c_i - c_j)(d_j - d_i)^{-1} \bmod n$$

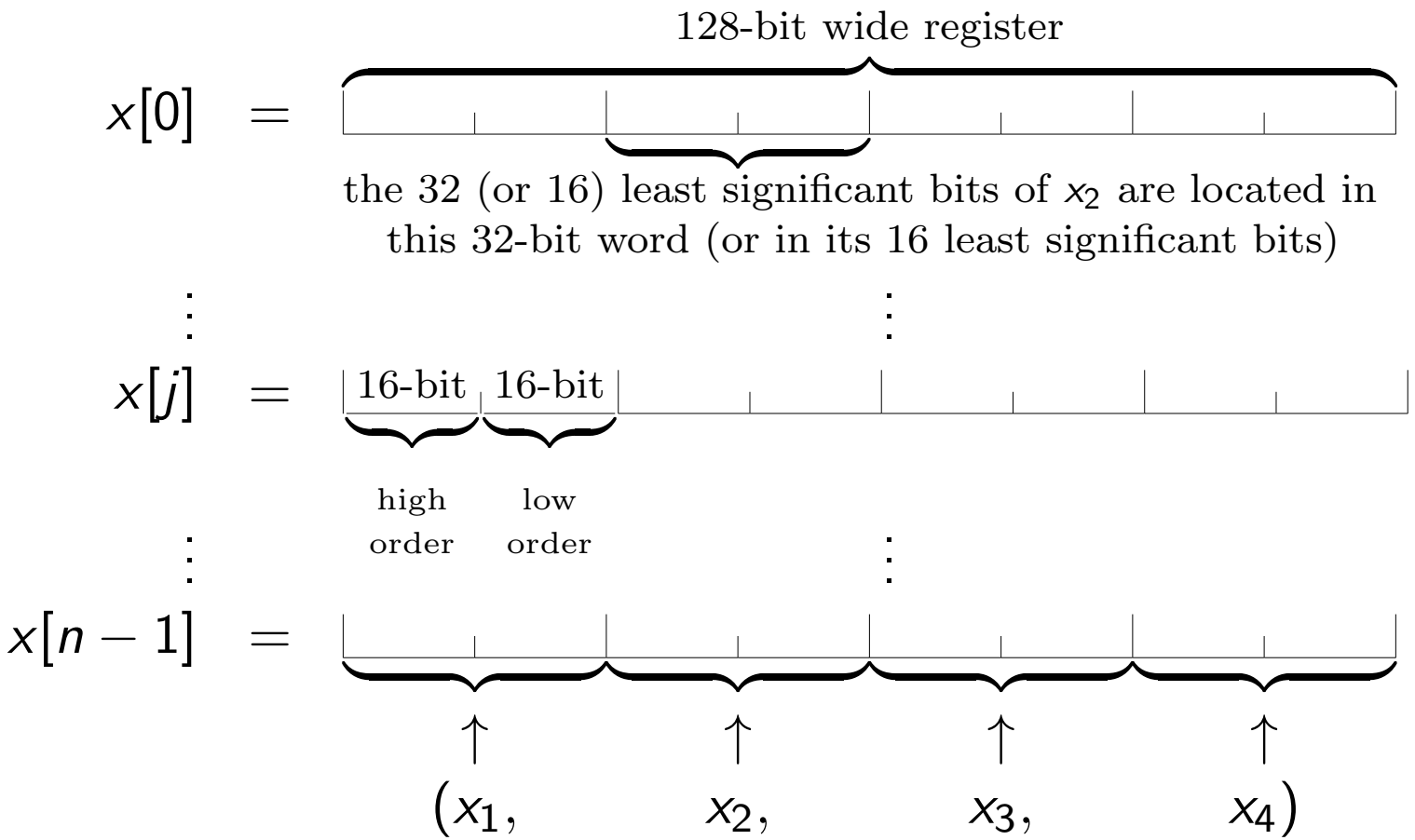
J. M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32:918-924, 1978.



Pollard Rho

- "Walk" through the set $\langle P \rangle$
- $X_i = c_i \cdot P + d_i \cdot Q$
- Iteration function $f : \langle P \rangle \to \langle P \rangle$
- This sequence eventually collides
- Expected number of steps (iterations): $\sqrt{\frac{\pi \cdot |\langle P \rangle|}{2}}$

Integer Representation



Implementation Details

- Optimize for high-throughput, not low-latency
  - Interleave two 4-way SIMD streams
- An efficient 4-way SIMD modular inversion algorithm
- Compute on 400 curves in parallel
  - simultaneous inversion (Montgomery)
- Do not use the negation map optimization

Implementation Details

- Optimize for high-throughput, not low-latency
  - Interleave two 4-way SIMD streams
- An efficient 4-way SIMD modular inversion algorithm
- Compute on 400 curves in parallel
  - simultaneous inversion (Montgomery)
- Do not use the negation map optimization

Trade correctness for speed

- When adding points X and Y do not check if $X = Y$.
  Save code size *and* increase performance (no branching).
- Faster modular reduction which might compute the wrong result.

Special Moduli

112-bit target

The 112-bit prime p used in the target curve $E(\mathbf{F}_p)$ is

$$p = \frac{2^{128}-3}{11 \cdot 6949}$$

Let $R = 2^{128}$, use a redundant representation modulo $\widetilde{p} = R - 3 = 11 \cdot 6949 \cdot p$

Note: $x \cdot 2^{128} \equiv x \cdot 3 \bmod \widetilde{p}$

$$\begin{array}{rccl} \mathfrak{R}: & \mathbf{Z}/2^{256}\mathbf{Z} & \rightarrow & \mathbf{Z}/2^{256}\mathbf{Z} \\ & x & \mapsto & \left(x \bmod 2^{128}\right) + 3 \cdot \left\lfloor \frac{x}{2^{128}} \right\rfloor \end{array}$$

$$x = x_H \cdot 2^{128} + x_L \equiv x_L + 3 \cdot x_H = \mathfrak{R}(x) \bmod \widetilde{p}$$

Sloppy Reduction

How often does it happen that $\mathfrak{R}(\mathfrak{R}(a \cdot b)) >= R$?

Given $x = x_0 + x_1 R$, $0 \leq x < R^2$, then
$\mathfrak{R}(x) = x_0 + 3x_1 = y = y_0 + y_1 R \leq 4R - 4$ and hence: $y_1 \leq 3$

Sloppy Reduction

How often does it happen that $\mathfrak{R}(\mathfrak{R}(a \cdot b)) >= R$?

Given $x = x_0 + x_1 R$, $0 \leq x < R^2$, then
$\mathfrak{R}(x) = x_0 + 3x_1 = y = y_0 + y_1 R \leq 4R - 4$ and hence: $y_1 \leq 3$

If $y_1 = 3$, then $y_0 + y_1 R = y_0 + 3R \leq 4R - 4$ and thus $y_0 \leq R - 4$.
If $y_1 \leq 2$, then $y_0 \leq R - 1$.

$$\mathfrak{R}(\mathfrak{R}(x)) = \left\{ \begin{array}{l} y_0 + 3y_1 \leq (R-4) + 3 \cdot 3 \\ y_0 + 2y_1 \leq (R-1) + 3 \cdot 2 \end{array} \right\} = R + 5.$$

Rough heuristic approximation: $\frac{6}{R+6}$

Sloppy Reduction

How often does it happen that $\mathfrak{R}(\mathfrak{R}(a \cdot b)) >= R$?

Given $x = x_0 + x_1 R$, $0 \leq x < R^2$, then
$\mathfrak{R}(x) = x_0 + 3x_1 = y = y_0 + y_1 R \leq 4R - 4$ and hence: $y_1 \leq 3$

If $y_1 = 3$, then $y_0 + y_1 R = y_0 + 3R \leq 4R - 4$ and thus $y_0 \leq R - 4$.
If $y_1 \leq 2$, then $y_0 \leq R - 1$.

$$\mathfrak{R}(\mathfrak{R}(x)) = \left\{ \begin{array}{l} y_0 + 3y_1 \leq (R-4) + 3 \cdot 3 \\ y_0 + 2y_1 \leq (R-1) + 3 \cdot 2 \end{array} \right\} = R + 5.$$

Rough heuristic approximation: $\frac{6}{R+6}$
More sophisticated heuristic:

$$\left(\frac{\varphi(\tilde{p})}{\tilde{p}}\right) \cdot \sum_{k=1,2} \left(3 - k - k \log\left(\frac{3}{k}\right)\right) \approx \frac{0.99118}{R} < \frac{1}{R}$$

Performance Results

Operation (sloppy modulus $\tilde{p} = 2^{128} - 3$, modulus $p = \frac{\tilde{p}}{11 \cdot 6949}$)	Average # cycles per two interleaved 4-SIMD operations	Average # cycles per operation	Operations per iteration	Average # cycles per iteration
Sloppy multiplication modulo $\tilde{p}$ (multiplication+reduction)	430 ($318 + 112$)	54 ($40 + 14$)	6	322
Modular subtraction	40 even, 24 odd: 40 total	5	6	30
Modular inversion	n/a	4941	$\frac{1}{400}$	12
Unique representation mod p	192	24	1	24
Miscellaneous	544	68	1	68
Total				456

Performance Results

Operation (sloppy modulus $\tilde{p} = 2^{128} - 3$, modulus $p = \frac{\tilde{p}}{11 \cdot 6949}$)	Average # cycles per two interleaved 4-SIMD operations	Average # cycles per operation	Operations per iteration	Average # cycles per iteration
Sloppy multiplication modulo $\tilde{p}$ (multiplication+reduction)	430 $(318 + 112)$	54 $(40 + 14)$	6	322
Modular subtraction	40 even, 24 odd: 40 total	5	6	30
Modular inversion	n/a	4941	$\frac{1}{400}$	12
Unique representation mod p	192	24	1	24
Miscellaneous	544	68	1	68
Total				456

Hence, our 214-PS3 cluster:

- computes $9.1 \cdot 10^9 \approx 2^{33}$ iterations per second
- works on $> 0.5M$ curves in parallel

Storage

- Per PS3: one distinguished point (4×16 bytes) per two second
- When storing the data naively: $\approx$ 300GB expected

Comparison

XC3S1000 FPGAs [1]

- FPGA-results of EC over 96- and 128-bit generic prime fields for COPACOBANA [2]
- Can host up to 120 FPGAs (US$ 10,000)

Our implementation

- Targeted at 112-bit prime curve
- Use 128-bit multiplication + fast reduction modulo $\widetilde{p}$
- For US$ 10,000 buy 33 PS3s

[1] T. Güneysu, C. Paar, and J. Pelzl. Special-purpose hardware for solving the elliptic curve discrete logarithm problem. *ACM Transactions on Reconfigurable Technology and Systems*, 1(2):1-21, 2008.
[2] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking ciphers with COPACOBANA a cost-optimized parallel code breaker. In CHES 2006, volume 4249 of LNCS, pages 101-118, 2006.

Comparison

	96 bits	128 bits
COPACOBANA	$4.0 \cdot 10^7$	$2.1 \cdot 10^7$
+ Moore's law	$7.9 \cdot 10^7$	$4.2 \cdot 10^7$
+ Negation map	$1.1 \cdot 10^8$	$5.9 \cdot 10^7$
PS3	$4.2 \cdot 10^7$	
33 PS3	$1.4 \cdot 10^9$	

33 PS3 / COPACOBANA (96 bits): 12.4 times faster
33 PS3 / COPACOBANA (128 bits): 23.8 times faster

Comparison

	96 bits	128 bits
COPACOBANA	$4.0 \cdot 10^7$	$2.1 \cdot 10^7$
+ Moore's law	$7.9 \cdot 10^7$	$4.2 \cdot 10^7$
+ Negation map	$1.1 \cdot 10^8$	$5.9 \cdot 10^7$
PS3	$4.2 \cdot 10^7$	
33 PS3	$1.4 \cdot 10^9$	

33 PS3 / COPACOBANA (96 bits): 12.4 times faster
33 PS3 / COPACOBANA (128 bits): 23.8 times faster

Note

The 33 dual-threaded PPE were not used

The new COPACOBANA has faster FPGAs
(no performance results known yet).

The 112-bit Solution

The point P of prime order n is given in the standard.
The x-coordinate of Q was chosen as $\lfloor (\pi - 3)10^{34} \rfloor$.

The 112-bit Solution

The point P of prime order n is given in the standard.
The x-coordinate of Q was chosen as $\lfloor(\pi - 3)10^{34}\rfloor$.

- Expected #iterations $\sqrt{\frac{\pi \cdot n}{2}} \approx 8.4 \cdot 10^{16}$
- January 13, 2009 – July 8, 2009 (not running continuously)
- When run continuously using the latest version of our code, the same calculation would have taken 3.5 months

$$
\begin{aligned}
P &= (1882814650579725348922237787137 52, \quad 3419875491033170827167861896082688)\\
Q &= (1415926535897932384626433832795028, \quad 3846759606494706724286139623885544)\\
n &= 4451685225093714776491891542548933
\end{aligned}
$$

The 112-bit Solution

The point P of prime order n is given in the standard.
The x-coordinate of Q was chosen as $\lfloor (\pi - 3)10^{34} \rfloor$.

- Expected #iterations $\sqrt{\frac{\pi \cdot n}{2}} \approx 8.4 \cdot 10^{16}$
- January 13, 2009 – July 8, 2009 (not running continuously)
- When run continuously using the latest version of our code, the same calculation would have taken 3.5 months

$$
\begin{aligned}
P = &\quad (1882814650579725348922237787137 52, \quad 3419875491033170827167861896082688) \\
Q = &\quad (1415926535897932384626433832795028, \quad 3846759606494706724286139623885544) \\
n = &\quad 4451685225093714776491891542548933
\end{aligned}
$$

$$Q = 312521636014772477161767351856699 \cdot P$$

Outline

- The Cell Broadband Engine Architecture
- Project 1: 112-bit prime field ECDLP
- Project 2: On the Use of the Negation Map in the Pollard Rho Method
  - *Joppe W. Bos, Thorsten Kleinjung, Arjen K. Lenstra: On the Use of the Negation Map in the Pollard Rho Method* In Algorithmic Number Theory (ANTS) 2010, volume 6197 of LNCS, pages 67-83, 2010

Motivation

Study the negation map in practice when solving the elliptic curve discrete logarithm problem over prime fields.

- The Suite B Cryptography by the NSA allows elliptic curves over prime fields only.
- Solve ECDLPs fast $\rightarrow$ break ECC-based schemes.

Using the (parallelized) Pollard ρ method

- 79-, 89-, 97- and 109-bit (2000) prime field Certicom challenges
- the 112-bit prime field ECDLP

have been solved.

Textbook optimization: negation map ($\sqrt{2}$ speed-up)

Motivation

Study the negation map in practice when solving the elliptic curve discrete logarithm problem over prime fields.

- The Suite B Cryptography by the NSA allows elliptic curves over prime fields only.
- Solve ECDLPs fast $\rightarrow$ break ECC-based schemes.

Using the (parallelized) Pollard ρ method

- 79-, 89-, 97- and 109-bit (2000) prime field Certicom challenges
- the 112-bit prime field ECDLP

have been solved.

Textbook optimization: negation map ($\sqrt{2}$ speed-up)
not used in any of the prime ECDLP records

Pollard ρ, [Pollard-78]

Approximate random walk in $\langle P \rangle$.

Index function $\ell : \langle P \rangle = G_0 \cup \ldots \cup G_{t-1} \mapsto [0, t-1]$

$$G_i = \{x : x \in \langle P \rangle, \ell(x) = i\}, \qquad |G_i| \approx \frac{n}{t}$$

Precomputed partition constants: $\mathfrak{f}_0, \ldots, \mathfrak{f}_{t-1}$

r-**adding walk**	$r+s$-**mixed walk**
$t = r$	$t = r + s$
$\mathfrak{p}_{i+1} = \mathfrak{p}_i + \mathfrak{f}_{\ell(\mathfrak{p}_i)}$	$\mathfrak{p}_{i+1} = \begin{cases} \mathfrak{p}_i + \mathfrak{f}_{\ell(\mathfrak{p}_i)}, & \text{if } 0 \le \ell(\mathfrak{p}_i) < r \\ 2\mathfrak{p}_i, & \text{if } \ell(\mathfrak{p}_i) \ge r \end{cases}$

[Teske-01]: r=20 performance close to a random walk.

The Negation Map

[Wiener,Zuccherato-98]

Equivalence relation $\sim$ on $\langle P \rangle$ by $\mathfrak{p} \sim -\mathfrak{p}$ for $\mathfrak{p} \in \langle P \rangle$.

$$\langle P \rangle \text{ of size } n \qquad \text{versus} \qquad \langle P \rangle / \sim \text{ of size about } \frac{n}{2}.$$

Advantage: Reduces the number of steps by a factor of $\sqrt{2}$.
Efficient to compute: Given $(x, y) \in \langle P \rangle \to -(x, y) = (x, -y)$

The Negation Map

[Wiener,Zuccherato-98]

Equivalence relation $\sim$ on $\langle P \rangle$ by $\mathfrak{p} \sim -\mathfrak{p}$ for $\mathfrak{p} \in \langle P \rangle$.

$$\langle P \rangle \text{ of size } n \qquad \text{versus} \qquad \langle P \rangle / \sim \text{ of size about } \tfrac{n}{2}.$$

Advantage: Reduces the number of steps by a factor of $\sqrt{2}$.
Efficient to compute: Given $(x, y) \in \langle P \rangle \to -(x, y) = (x, -y)$

Compute $\begin{matrix} \mathfrak{p}_i + \mathfrak{f}_{\ell(\mathfrak{p}_i)} \\ -(\mathfrak{p}_i + \mathfrak{f}_{\ell(\mathfrak{p}_i)}) \end{matrix}$

The Negation Map

[Wiener,Zuccherato-98]

Equivalence relation $\sim$ on $\langle P \rangle$ by $\mathfrak{p} \sim -\mathfrak{p}$ for $\mathfrak{p} \in \langle P \rangle$.

$$\langle P \rangle \text{ of size } n \qquad \text{versus} \qquad \langle P \rangle / \sim \text{ of size about } \tfrac{n}{2}.$$

Advantage: Reduces the number of steps by a factor of $\sqrt{2}$.
Efficient to compute: Given $(x, y) \in \langle P \rangle \to -(x, y) = (x, -y)$

Compute
$$\begin{aligned} &\mathfrak{p}_i + \mathfrak{f}_{\ell(\mathfrak{p}_i)} \\ &-(\mathfrak{p}_i + \mathfrak{f}_{\ell(\mathfrak{p}_i)}) \quad = \mathfrak{p}_{i+1} \end{aligned}$$

Negation Map, Side-Effects

Well-known disadvantage: as presented no solution to large ECDLPs

Negation Map, Side-Effects

Well-known disadvantage: fruitless cycles

$$\mathfrak{p} \xrightarrow{(i,-)} -(\mathfrak{p} + \mathfrak{f}_i) \xrightarrow{(i,-)} \mathfrak{p}.$$

Fruitless 2-cycle starts from a random point with probability $\frac{1}{2r}$
[Duursma,Gaudry,Morain-99] (Proposition 31)

Negation Map, Side-Effects

Well-known disadvantage: fruitless cycles

$$\mathfrak{p} \xrightarrow{(i,-)} -(\mathfrak{p} + \mathfrak{f}_i) \xrightarrow{(i,-)} \mathfrak{p}.$$

Fruitless 2-cycle starts from a random point with probability $\frac{1}{2r}$
[Duursma,Gaudry,Morain-99] (Proposition 31)

2-cycle reduction technique: [Wiener,Zuccherato-98]

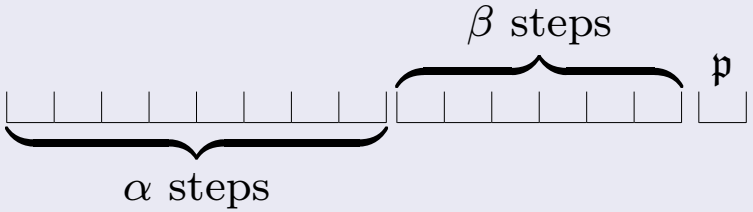
$$f(\mathfrak{p}) = \begin{cases} E(\mathfrak{p}) & \text{if } j = \ell(\sim(\mathfrak{p} + \mathfrak{f}_j)) \text{ for } 0 \leq j < r \\ \sim(\mathfrak{p} + \mathfrak{f}_i) & \text{with } i \geq \ell(\mathfrak{p}) \text{ minimal s.t. } \ell(\sim(\mathfrak{p} + \mathfrak{f}_i)) \neq i \bmod r. \end{cases}$$

once every r^r steps: $E : \langle P \rangle \to \langle P \rangle$ may restart the walk

Cost increase $c = \sum_{i=0}^{r} \frac{1}{r^i}$ with $1 + \frac{1}{r} \leq c \leq 1 + \frac{1}{r-1}$.

Dealing with Fruitless Cycles in General [Gallant,Lambert,Vanstone-00]

Cycle detection



Compare $\mathfrak{p}$ to all β points. Detect cycles of length $\leq \beta$.

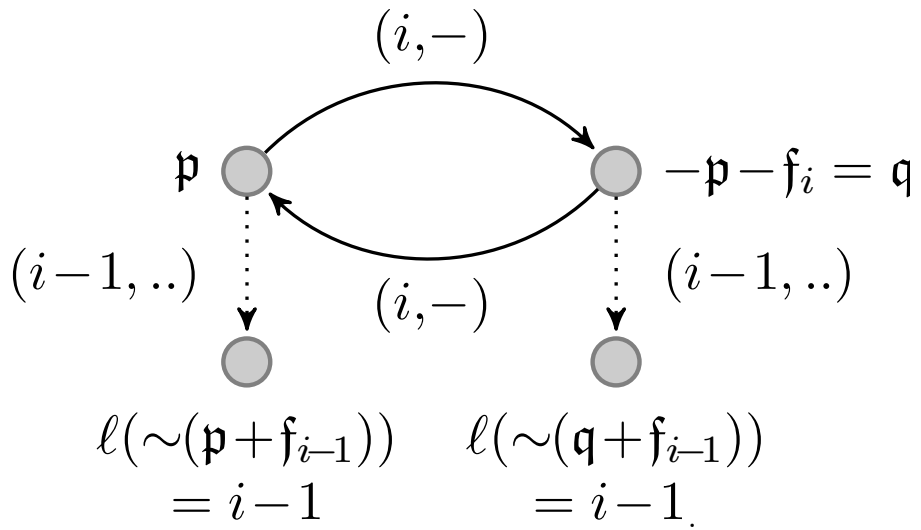
Cycle Escaping

Add

- $\mathfrak{f}_{\ell(\mathfrak{p})+c}$ for a fixed $c \in \mathbf{Z}$
- a precomputed value $\mathfrak{f}'$
- $\mathfrak{f}''_{\ell(\mathfrak{p})}$ from a distinct list of r precomputed values $\mathfrak{f}''_0, \mathfrak{f}''_1, \ldots, \mathfrak{f}''_{r-1}$.

to a representative element of this cycle.

2-cycles when using the 2-cycle reduction technique



Lemma

The probability to enter a fruitless 2-cycle when looking ahead to reduce 2-cycles while using an r-adding walk is

$$\frac{1}{2r}\left(\sum_{i=1}^{r-1}\frac{1}{r^i}\right)^2=\frac{(r^{r-1}-1)^2}{2r^{2r-1}(r-1)^2}=\frac{1}{2r^3}+O\left(\frac{1}{r^4}\right).$$

4-cycle Reduction

$$\mathfrak{p} \xrightarrow{(i,+)} \mathfrak{p} + \mathfrak{f}_i \xrightarrow{(j,-)} -\mathfrak{p} - \mathfrak{f}_i - \mathfrak{f}_j \xrightarrow{(i,+)} -\mathfrak{p} - \mathfrak{f}_j \xrightarrow{(j,-)} \mathfrak{p}.$$

Fruitless 4-cycle starts with probability $\frac{r-1}{4r^3}$.

4-cycle Reduction

$$\mathfrak{p} \xrightarrow{(i,+)} \mathfrak{p}+\mathfrak{f}_i \xrightarrow{(j,-)} -\mathfrak{p}-\mathfrak{f}_i-\mathfrak{f}_j \xrightarrow{(i,+)} -\mathfrak{p}-\mathfrak{f}_j \xrightarrow{(j,-)} \mathfrak{p}.$$

Fruitless 4-cycle starts with probability $\frac{r-1}{4r^3}$.
Extend the 2-cycle reduction method to reduce 4-cycles:

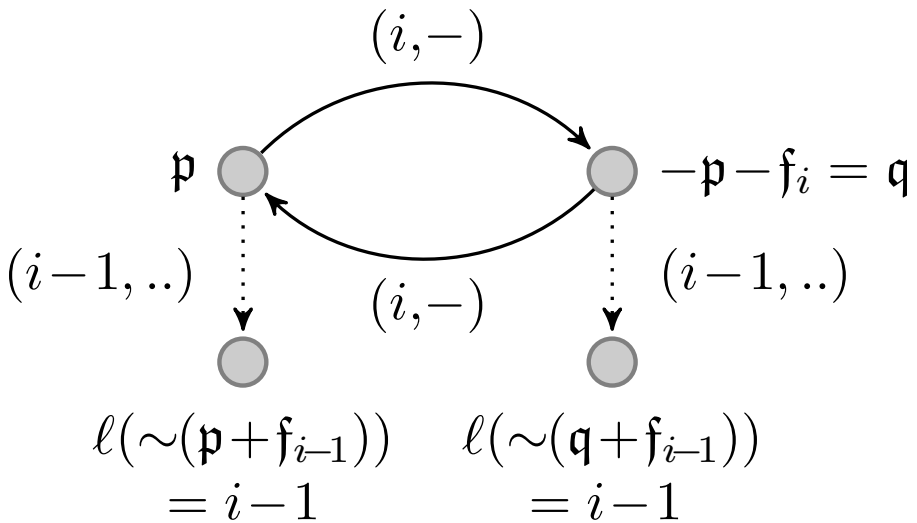
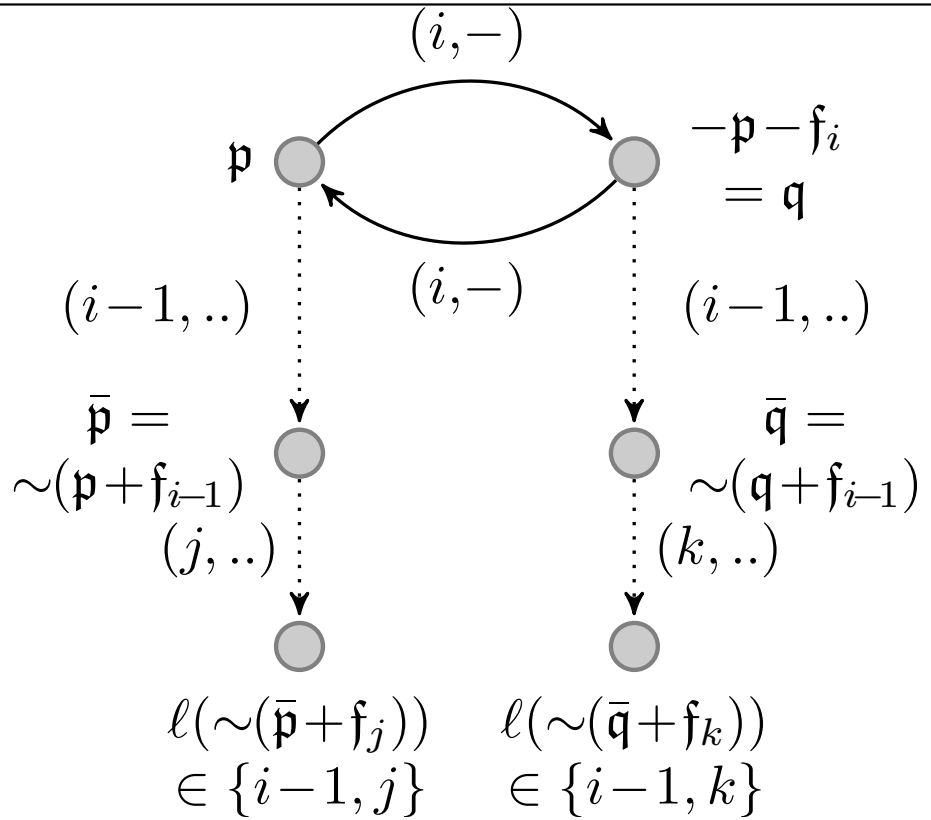
$$g(\mathfrak{p})=\begin{cases} E(\mathfrak{p}) & \text{if } j \in \{\ell(\mathfrak{q}), \ell(\sim(\mathfrak{q}+\mathfrak{f}_{\ell(\mathfrak{q})}))\} \text{ or } \ell(\mathfrak{q}) = \ell(\sim(\mathfrak{q}+\mathfrak{f}_{\ell(\mathfrak{q})})) \\ & \text{where } \mathfrak{q} = \sim(\mathfrak{p}+\mathfrak{f}_j), \text{ for } 0 \le j < r, \\ \mathfrak{q} = \sim(\mathfrak{p}+\mathfrak{f}_i) & \text{with } i \ge \ell(\mathfrak{p}) \text{ minimal s.t.} \\ & i \bmod r \neq \ell(\mathfrak{q}) \neq \ell(\sim(\mathfrak{q}+\mathfrak{f}_{\ell(\mathfrak{q})})) \neq i \bmod r. \end{cases}$$

Disadvantage: more expensive iteration function: $\ge \frac{r+4}{r}$

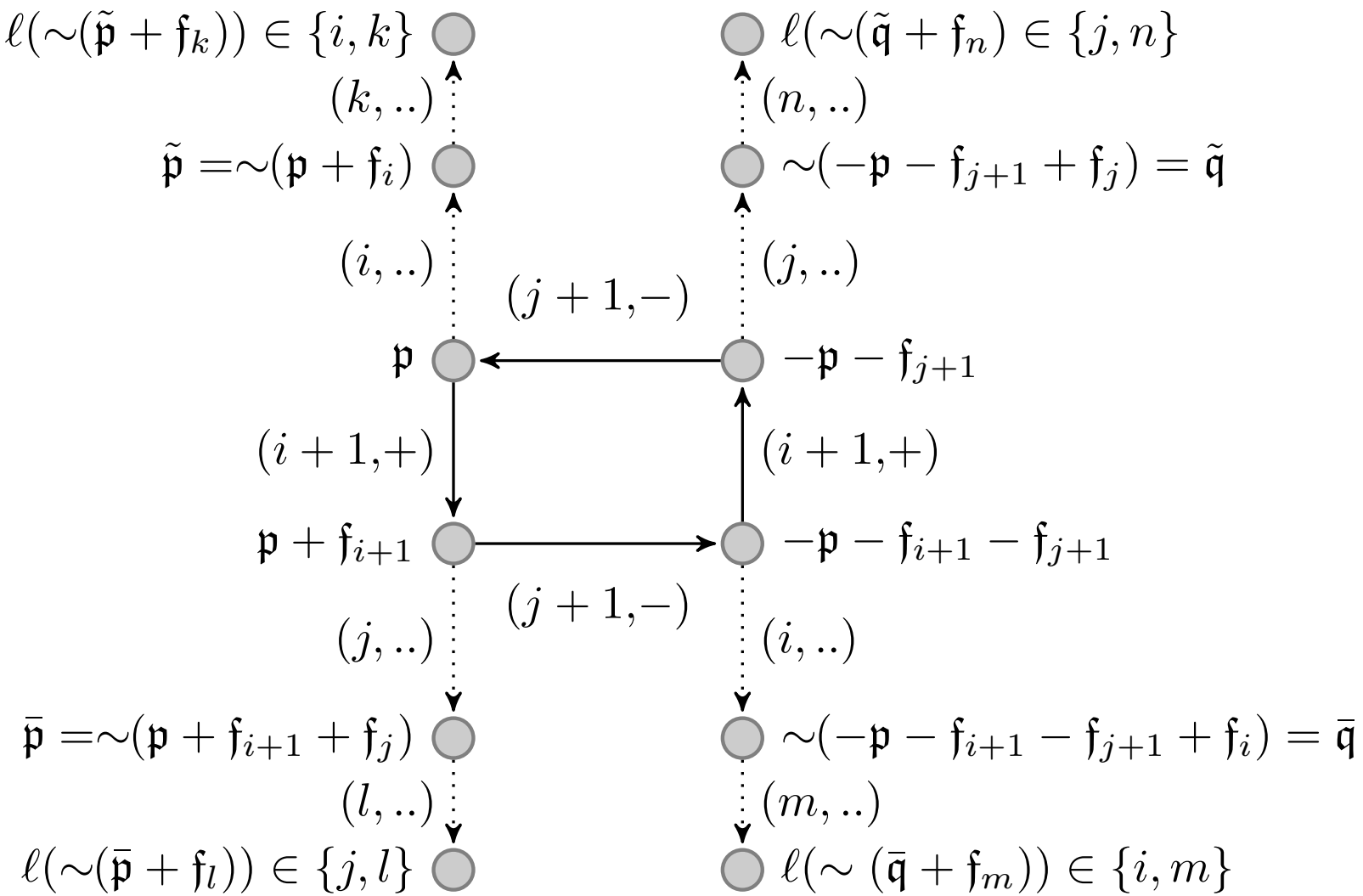
Advantage: positive effect of $\sqrt{\frac{r-1}{r}}$ since

$$\text{image}(g) \subset \langle P \rangle \text{ with } |\text{image}(g)| \approx \tfrac{r-1}{r}|\langle P \rangle|.$$

2-cycles with Cycle Reduction

With 2-cycle reduction	With 4-cycle reduction
	
$\geq \frac{1}{2r^3}$	$\geq \frac{2(r-2)^2}{(r-1)r^4}$

Example: 4-cycle with 4-cycle reduction

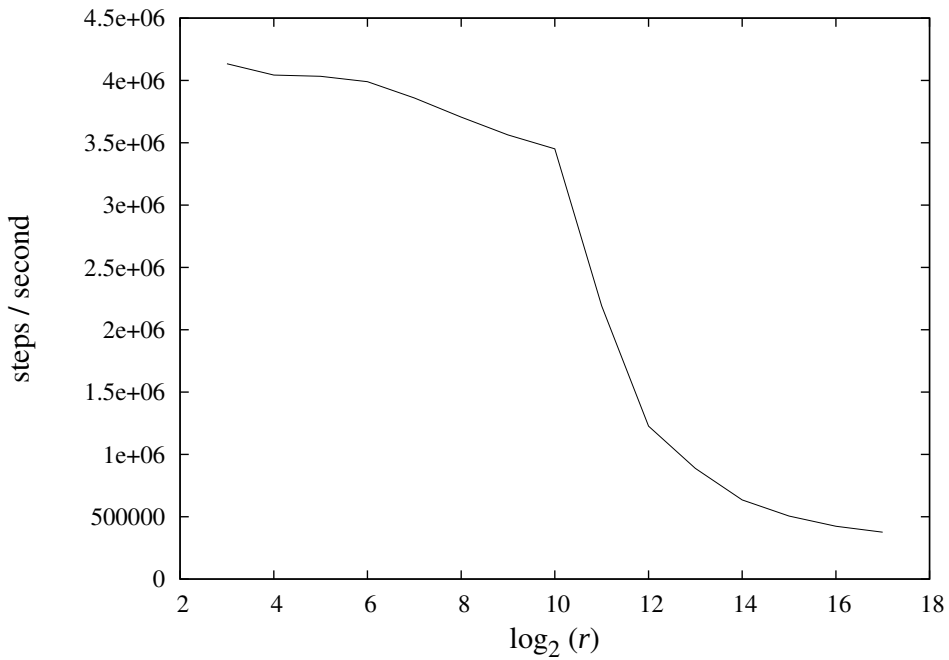


Size of the Random Walk

- Probability to enter cycle depends on the number of partitions r
- Why not simply increase r?

Size of the Random Walk

- Probability to enter cycle depends on the number of partitions r
- Why not simply increase r?



- Practical performance penalty (cache-misses)
- Fruitless cycles still occur

Recurring Cycles

Using

- r-adding walk with a medium sized r **and**
- $\{ 2, 4 \}$-reduction technique **and**
- cycle escaping techniques

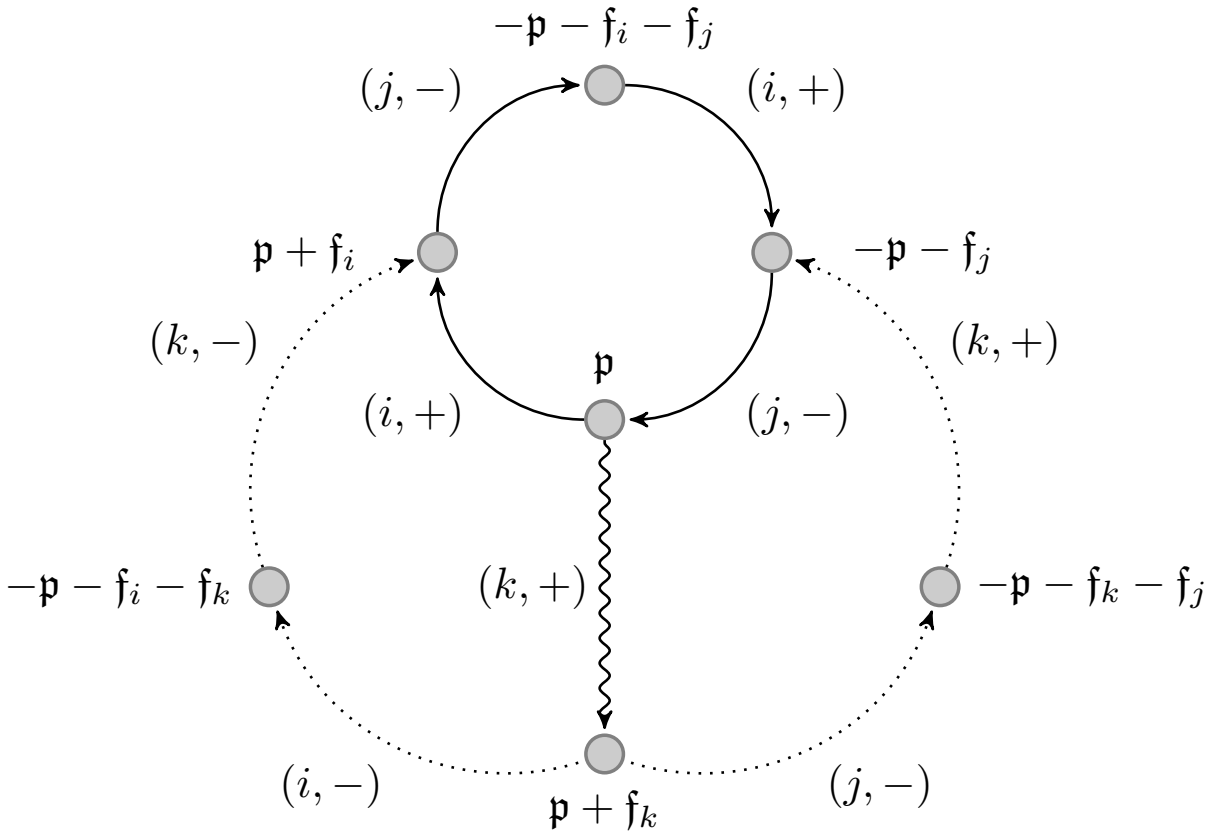
it is expected that many walks will never find a DTP.

Recurring Cycles

Using

- r-adding walk with a medium sized r **and**
- $\{ 2, 4 \}$-reduction technique **and**
- cycle escaping techniques

it is expected that many walks will never find a DTP.



Probabilities Overview

Cycle reduction method:		none	2-cycle	4-cycle
Probability to enter	2-cycle	$\mathbf{\frac{1}{2r}}$	$\frac{1}{2r^3}$	$\frac{2(r-2)^2}{(r-1)r^4}$
	4-cycle	$\mathbf{\frac{r-1}{4r^3}}$	$\frac{r-1}{4r^3}$	$\frac{4(r-2)^4(r-1)}{r^{11}}$
Probability to recur to escape point using	$\mathfrak{f}_{\ell(\mathfrak{p})+c}$	$\frac{1}{2r}$	$\frac{1}{2r^2}$	$\frac{(r-2)^2}{r^4}$
	$\mathfrak{f}'$	$\frac{1}{8r}$	$\frac{1}{8r^3}$	$\frac{(r-2)^2}{2r^5}$
	$\mathfrak{f}''_{\ell(\mathfrak{p})}$	$\frac{1}{8r^2}$	$\frac{1}{8r^4}$	$\frac{(r-2)^2}{2r^6}$
Slowdown factor of iteration function		n/a	$\frac{r+1}{r}$	$\frac{r+4}{r}$

Dealing with Recurring Cycles

Heuristic

A cycle with at least one doubling is most likely not fruitless.

Reduce the number of fruitless (recurring) cycles by using a mixed-walk

- **Advantage**: Avoid recurring cycles
- **Disadvantage**: EC-doublings (7M) are more expensive than EC-additions (6M)

Experiments @ AMD Phenom 9500

Long-term yield: run 2×10^9 iterations, ignore the first 10^9.

Yield: speed-up

$$\left.\begin{array}{l}\#\text{additional additions} \\ \#\text{duplications}\end{array}\right\} \text{max. theoretical speedup}$$

	$r = 16$	$r = 32$	$r = 64$	$r = 128$	$r = 256$	$r = 512$
Without negation map						
	7.29: 0.98	7.28: 0.99	**7.27**: 1.00	7.19: 0.99	6.97: 0.96	6.78: 0.94
With negation map						
just g	0.00: 0.00	0.00: 0.00	0.00: 0.00	0.00: 0.00	0.04: 0.01	3.59: 0.70
just $\bar{e}$	3.34: 0.64	4.89: 0.95	5.85: 1.14	6.10: 1.19	6.28: 1.23	6.18: 1.21
f, e	0.00: 0.00 *9.4e8 / 0.0e0* } *0.08*	0.00: 0.00 *6.6e8 / 0.0e0* } *0.48*	1.52: 0.30 *1.0e8 / 0.0e0* } *1.28*	5.93: 1.16 *3.6e7 / 0.0e0* } *1.37*	6.47: 1.27 *2.9e7 / 0.0e0* } *1.38*	6.36: 1.25 *2.5e7 / 0.0e0* } *1.39*
f, $\bar{e}$	3.71: 0.72 *9.2e7 / 9.9e5* } *1.27*	6.36: 1.24 *6.8e7 / 2.8e5* } *1.32*	6.50: 1.27 *4.2e7 / 6.5e4* } *1.36*	6.57: **1.29** *3.3e7 / 1.5e4* } *1.38*	6.47: 1.27 *2.9e7 / 3.8e3* } *1.38*	6.30: 1.25 *2.7e7 / 9.7e2* } *1.39*
g, e	0.00: 0.00 *8.7e8 / 0.0e0* } *0.19*	0.01: 0.00 *3.7e8 / 0.0e0* } *0.91*	4.89: 0.96 *6.6e7 / 0.0e0* } *1.34*	6.22: 1.22 *4.2e7 / 0.0e0* } *1.37*	6.23: 1.22 *3.3e7 / 0.0e0* } *1.38*	6.05: 1.19 *1.3e7 / 0.0e0* } *1.41*
g, $\bar{e}$	0.76: 0.15 *3.3e8 / 1.6e5* } *0.97*	5.91: 1.17 *1.7e8 / 6.0e4* } *1.19*	6.02: 1.18 *8.1e7 / 8.1e3* } *1.32*	6.25: 1.23 *5.4e7 / 1.0e3* } *1.35*	6.13: 1.20 *4.0e7 / 1.2e2* } *1.37*	6.00: 1.18 *2.7e7 / 9.0e0* } *1.39*

Conclusions

Using the negation map optimization technique for solving prime ECDLPs is useful in practice when

- $\{ 2, 4 \}$-cycle reduction techniques are used
- recurring cycles are avoided; e.g. escaping by doubling
- use medium sized r-adding walk ($r = 128$)

Using all this we managed to get a speedup of at most:

$$\mathbf{1.29} < \sqrt{2} \ (\approx 1.41)$$

More details and experiments in the article.

Future Work

Better cycle reduction or escaping techniques?
Can we do better than 1.29 speedup?
Special algorithms for SIMD-architectures.

Conclusions

Future Work

Better cycle reduction or escaping techniques?
Can we do better than 1.29 speedup?
Special algorithms for SIMD-architectures.

D. J. Bernstein, T. Lange, and P. Schwabe: On the correct use of the negation map in the Pollard rho method. PKC 2011

- Straight-line algorithm to compute the negation map (branch-free)
- 2048-adding walk on the cache-less SPE of the Cell
- no direct comparison between negation and non-negation map setting
- estimated ≈ 1.37 speed-up