Pollard Rho on the PlayStation 3

Joppe W. Bos[1]
Marcelo E. Kaihara[1]
Peter L. Montgomery[2]

[1] EPFL IC LACAL, CH-1015 Lausanne, Switzerland
[2] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA

RAIM'09 October 27th 2009 LIP ENS Lyon

Motivation

- Elliptic Curve Cryptography (ECC):
  - Widely standardized
    - Standard for Efficient Cryptography 2, SEC2 (112-521 bit)
    - Wireless Transport Layer Security Specification (112-224 bit)
    - Digital Signature Standard, FIPS 186-3, NIST (192-521 bit)
  - Security relies on hardness of solving Elliptic Curve Discrete Logarithm Problem (ECDLP)

⬇

- Are the standardized key sizes secure?
- What is the practical cost of solving the ECDLP?

Objective

- Evaluate cost of solving ECDLP for small key sizes
- 112-bit ECC standard

- Use broadly available platform
- PlayStation 3:
  - Low price
  - Hybrid multi-core architecture

- Implement Pollard rho on the Cell architecture
  - Design SIMD arithmetic algorithms
  - Optimize modular arithmetic for 112-bit prime

ECDLP

- Settings:

  E is an elliptic curve over F_p with p odd prime

  $P \in E(F_p)$ is a point of order n

  $Q = k \cdot P \in \langle P \rangle$

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

$$k = \log_P Q$$

- Largest solved instance 109-bit prime field (2002)
- It took "10^4 computers (mostly PCs) running 24 hours a day for 549 days".

Solving the ECDLP

- Pollard rho:
  - The most efficient algorithm in the literature (for generic curves).
  - The underlying idea of this method is to search for two distinct pairs

$$(c_i, d_i), (c_j, d_j) \in Z/nZ \times Z/nZ \text{ such that}$$

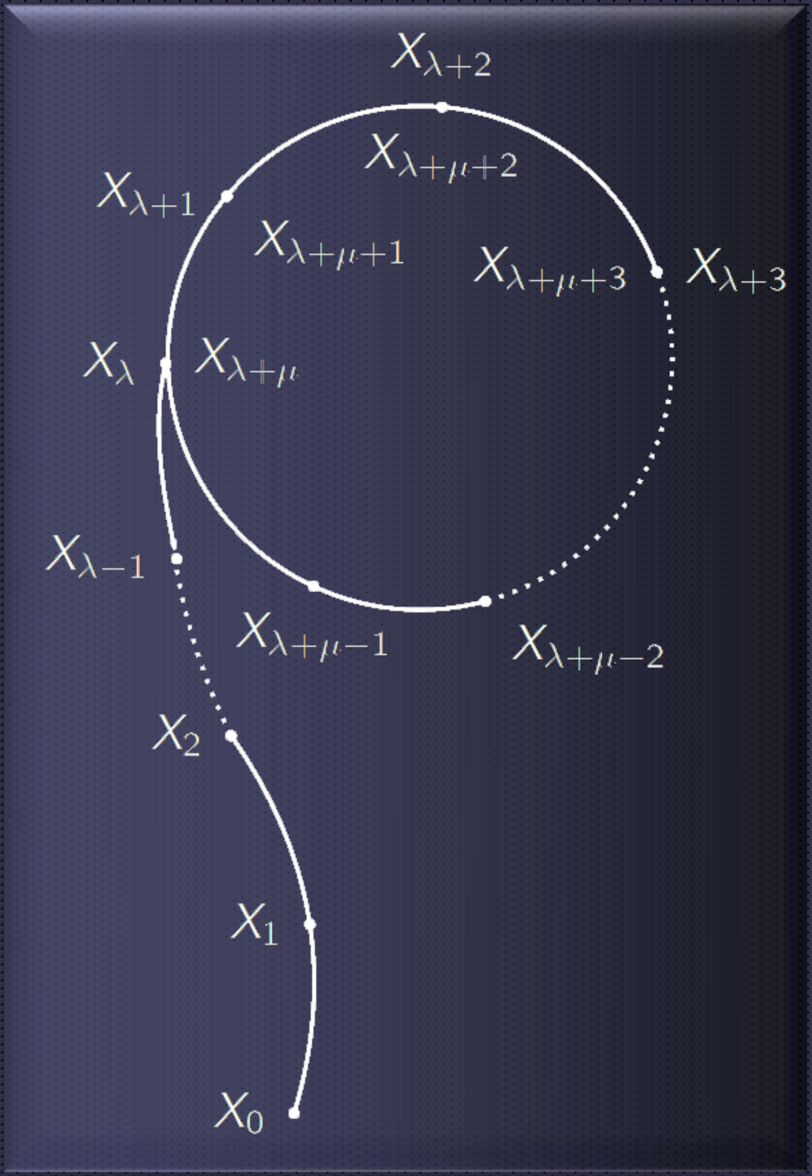
$$c_i \cdot P + d_i \cdot Q = c_j \cdot P + d_j \cdot Q$$

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

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

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

Pollard Rho



- "Walk" through the set $\langle P \rangle$

$$X_i = c_i \cdot P + d_i \cdot Q$$

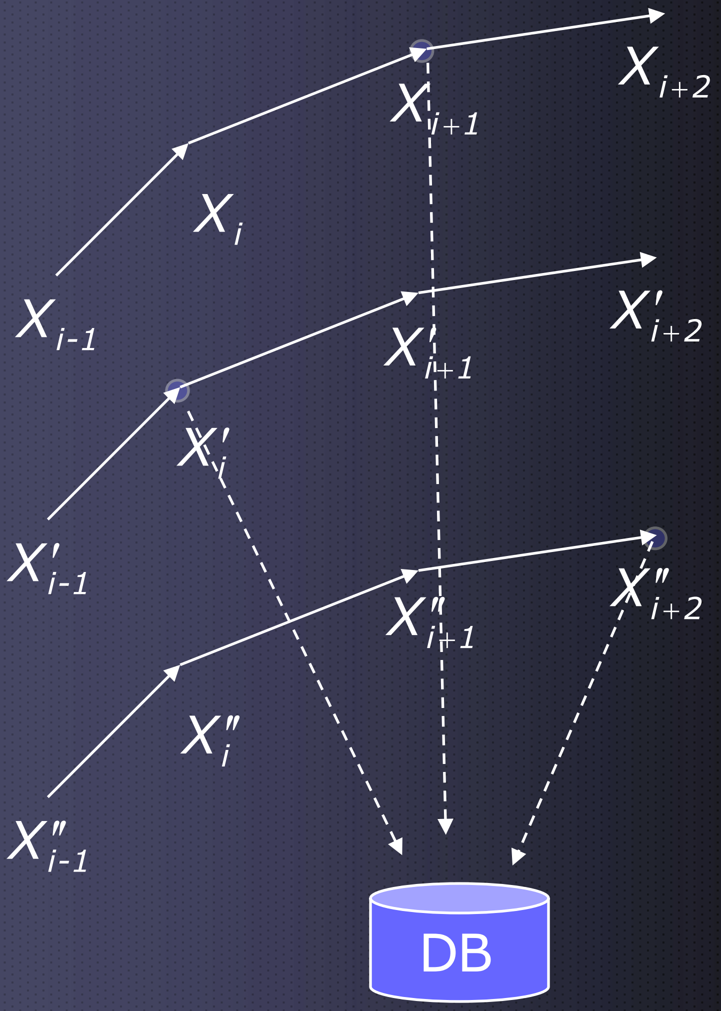
- Iteration function $f : \langle P \rangle \to \langle P \rangle$

$$X_{i+1} = f(X_i), \ \ i \geq 0$$

- This sequence eventually collides
- Expected number of iterations

$$\sqrt{\frac{\pi \cdot |\langle P \rangle|}{2}}$$

Optimization I



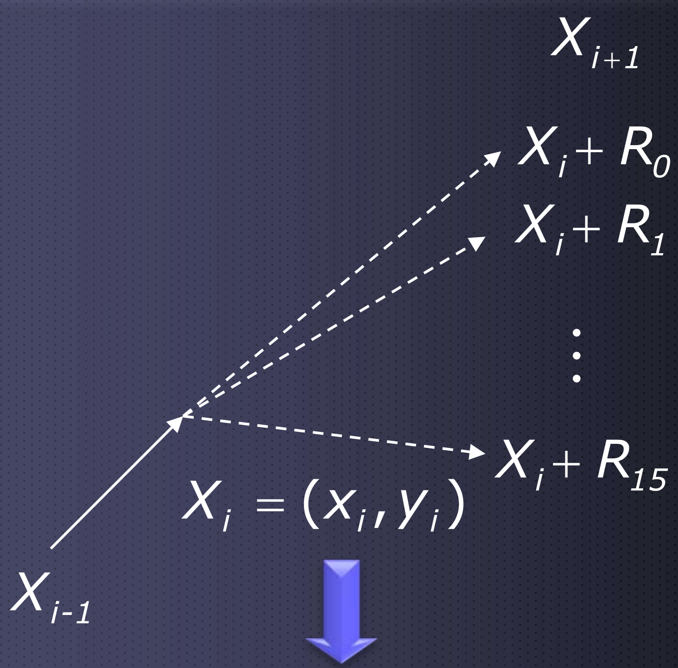
- Parallel version: distinguish points and send them to a central server

  P.C. van Oorschot and M. J. Wiener, [1999].

  - *Mark points with a certain property e.g., $X_i=(x_i,y_i)$, DPT: $2^{24} \mid x_i$*
  - *Communicate them to a central DB to check collisions*

- *Leads to a linear speed-up on the number of processors.*

Optimization II



Use the least significant
4-bit to determine
the next partition

- r-adding walks, *E. Teske*, [2001].
- Divide $\langle P \rangle$ into r different partitions

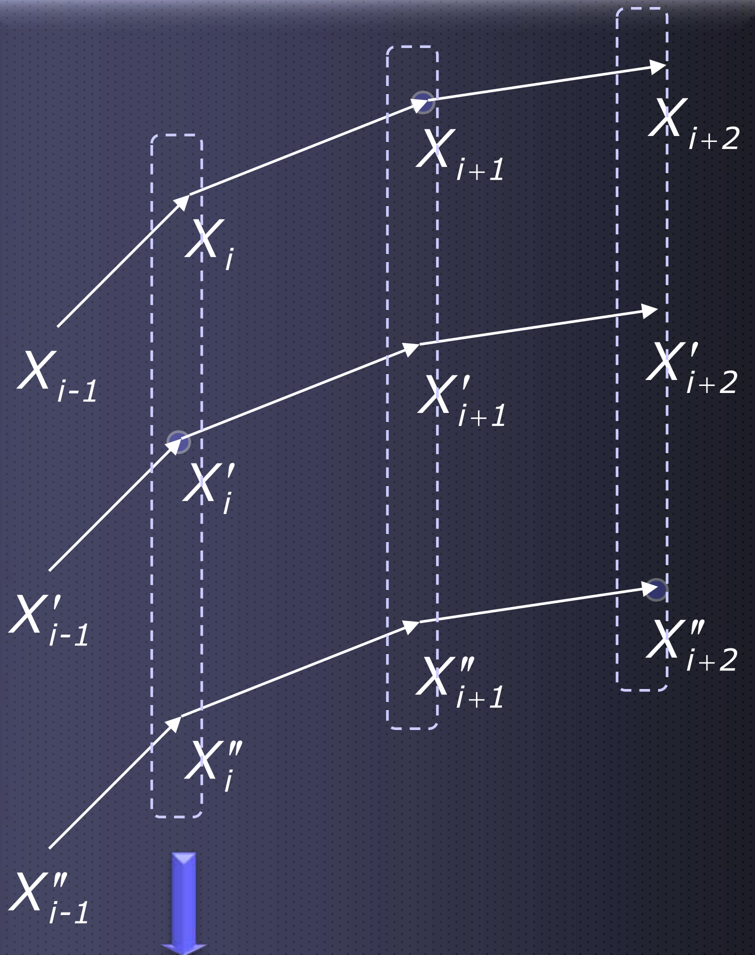
$$h : \langle P \rangle \rightarrow [0, r-1]$$

- For each partition: $R_j = c_j \cdot P + d_j \cdot Q$

$$X_{i+1} = f(X_i) = X_i + R_{h(X_i)}$$

$r \geq 16$ partitions $\approx$ random mapping

Optimization III



Apply to independent walks

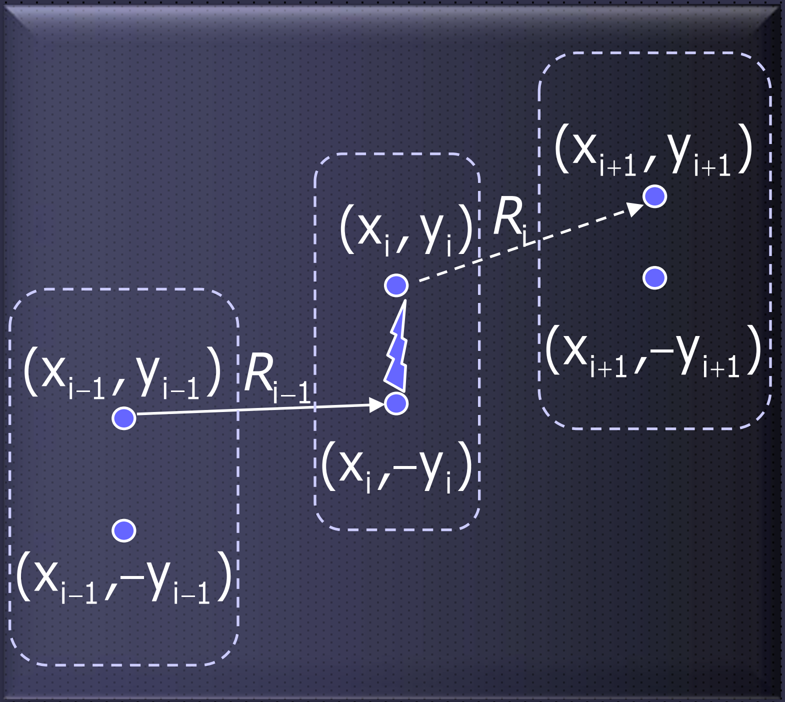
- Simultaneous Inversion, trade inversions for multiplications

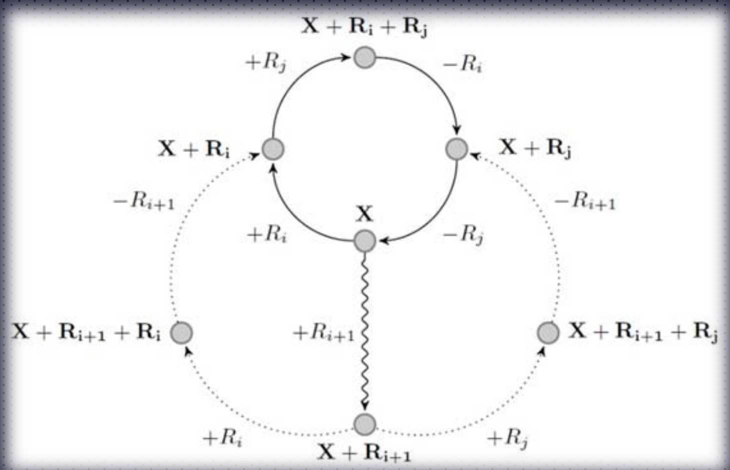
  P.L. Montgomery, [1987].

  - *Suitable for cryptanalytic purposes*
  - *Trade M modular inversions for 3(M-1) modular multiplications and 1 modular inversion*

Affine Weierstrass representation

Optimization IV





- **Negation Map (not used)**

  M.J. Wiener and R. J. Zuccherato, [1998].

  Computation of the negative is cheap

  $$-P = (x, -y)$$

  Given an equivalence relation ~ on $\langle P \rangle$
  Iterate over the set of equivalence classes

  $$\langle P \rangle / \sim$$

  Reduce search space by a factor of 2

The PlayStation 3

- The Cell contains
  - 1 “Power Processor Element ” (PPE)
  - 8 “Synergistic Processing Elements” (SPEs)

    (6 available to the user in the PS3 under Linux)

- Characteristics of the SPEs:
  - 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)

Programming Constraints

- *Memory*
  - The executable and all data should fit in the LS (256KB).
- *Branches*
  - No “smart” dynamic branch prediction.
  - Instead “prepare-to-branch” instructions to redirect instruction prefetch to branch targets.
- *Instruction set limitations*
  - 16 x 16 $\rightarrow$ 32 bit multipliers (4-SIMD)
- *Dual pipeline*
  - One odd and one even instruction can be dispatched per clock cycle.

Arithmetic

- Using affine Weierstrass representation

$$P, Q \in E(F_p) \setminus \{O\} \quad P = (x_1, y_1) \text{ and } Q = (x_2, y_2)$$

$$\text{If } P \neq Q \text{ then } P + Q = (x_3, y_3)$$

$$x_3 = \mu^2 - x_1 - x_2$$

$$y_3 = \mu(x_1 - x_3) - y_1$$

$$\mu = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \text{if } P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1} & \text{if } P = Q \end{cases}$$

Using Montgomery's simultaneous inversion and running M curves in parallel.

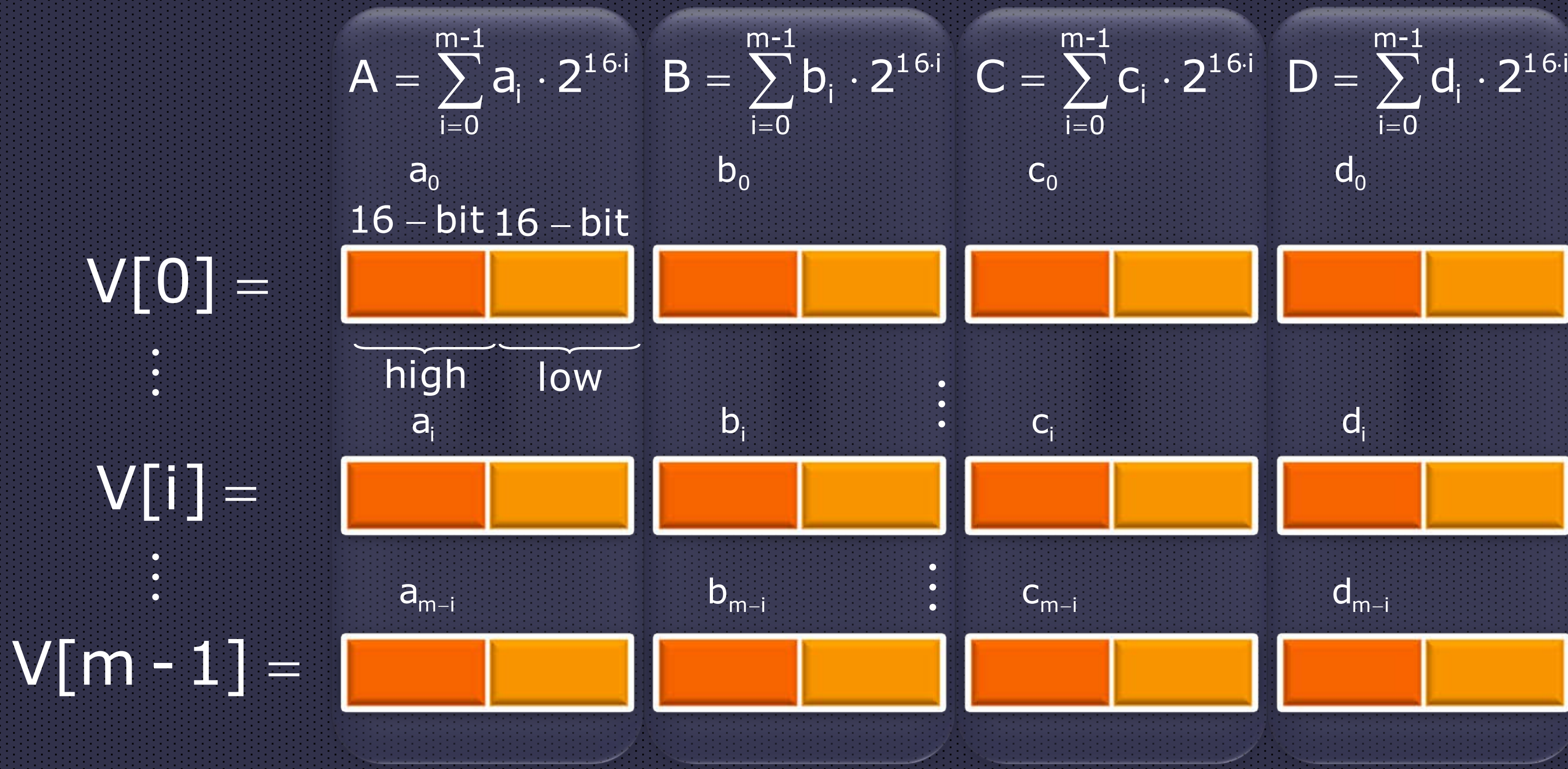
6 modular multiplications

6 modular subtractions

$\frac{1}{M}$ modular inversions

Integer Representation

Integers A, B, C, D represented in radix 2^{16}



4 - SIMD

If $0 \leq a; b; c; d < 2^{16}$, then $a \times b + c + d < 2^{32}$.

Use the multiply-and-add instruction and an extra addition of carries.

Branch-free implementation.

Modular Reduction

- The prime 112-bit p in the target curve $E(F_p)$ is

$$p = \mathrm{DB7C2ABF62E35E668076BEAD208B}_{16}$$

Modular Reduction

- The prime 112-bit p in the target curve $E(F_p)$ is

$$p = \mathrm{DB7C2ABF62E35E668076BEAD208B}_{16}$$

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

Modular Reduction

- The prime 112-bit p in the target curve $E(F_p)$ is

$$p = \mathrm{DB7C2ABF62E35E668076BEAD208B}_{16}$$

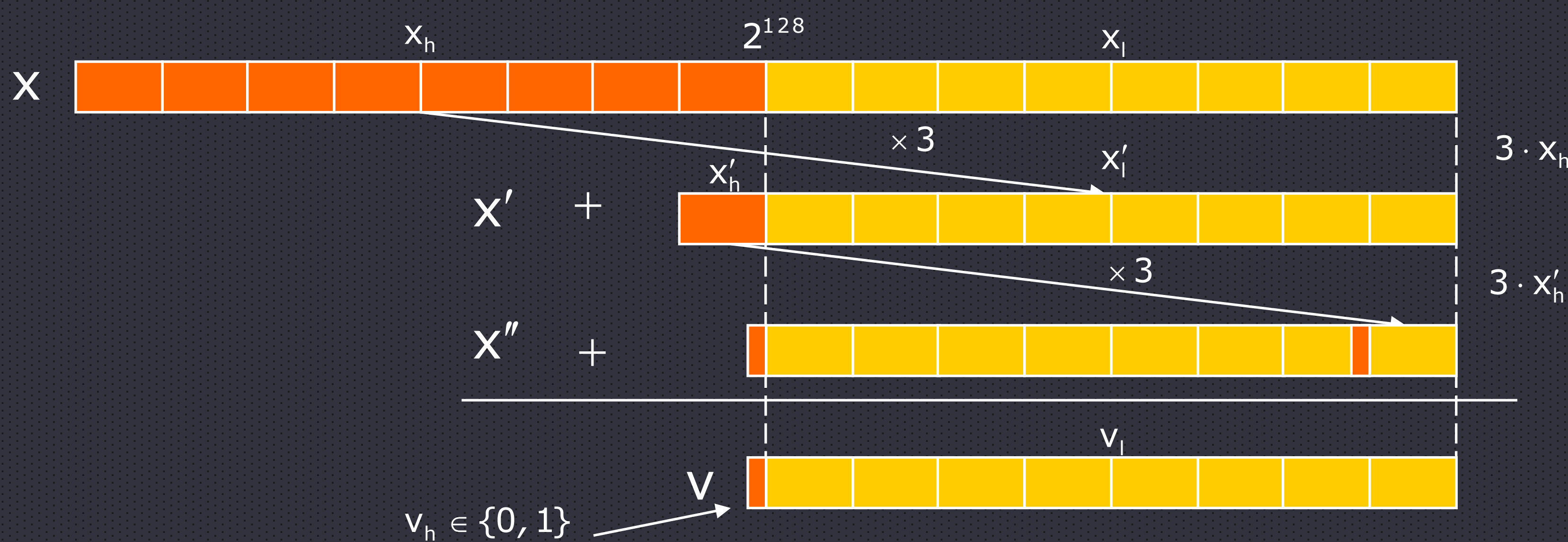
$$p = \frac{2^{128} - 3}{11 \cdot 6949}$$

- Perform calculation using a redundant representation

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

Fast reduction

Use modulus $\tilde{p} = 2^{128} - 3 = 11 \cdot 6949 \cdot p$



Overwhelming prob. $v_h = 0$

$$R : Z/2^{256}Z \to Z/2^{256}Z$$

$$x \to (x \bmod 2^{128}) + 3 \cdot \left\lfloor \frac{x}{2^{128}} \right\rfloor$$

$$x = x_H \cdot 2^{128} + x_L \equiv x_L + 3 \cdot x_H = R(x) \mod \tilde{p}$$

Fast Modular Multiplication

- *Proposition*

For independent random 128-bit non-negative integers x and y there is overwhelming probability that

$$0 \leq R(R(x \cdot y)) < \tilde{p}$$

Counter-examples easy to construct:

$$0 \leq R(R(x)) < 2^{128} + 6$$

During the whole run not a single faulty reduction

Distinguish Point Property

- Need to uniquely determine the partition number and DTP property during the r-adding walk.

$$P = (x, y)$$

$$\Downarrow$$

$$x : 0 \le x < \tilde{p}$$

- Partial Montgomery Reduction in order to reduce modulo p.

$$x' = x \cdot 2^{-16} \bmod p$$

- Check least significant 24 bits of *x* in partial Montgomery representation.

Modular Inversion

- *Based on Extended Binary GCD algorithm:* $z \equiv x^{-1} \bmod p$

$p \quad\quad 0$

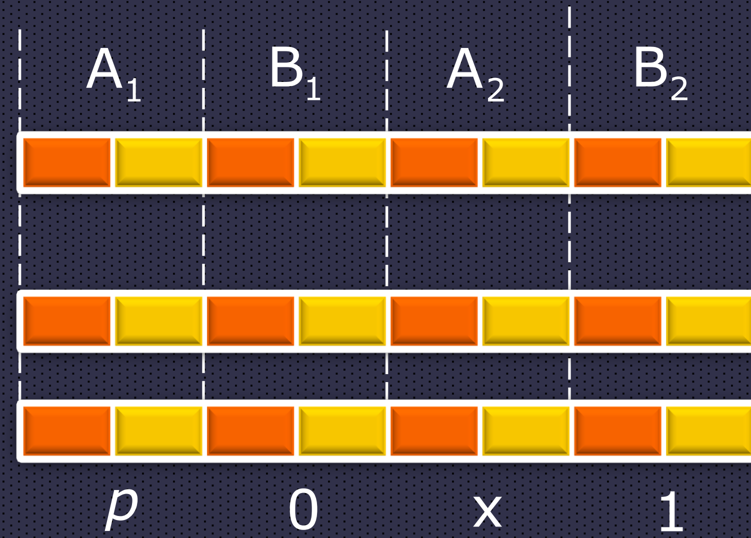
$\downarrow \quad\quad \downarrow$

$$A_1 \times 1 \equiv B_1 \times x \mod p$$

$$A_2 \times 1 \equiv B_2 \times x \mod p$$

$\uparrow \quad\quad \uparrow$

$x \quad\quad 1$



$r = 2^{32}$

Compute $gcd(A_1, A_2)$

Obtain from B_2 *almost Montgomery inverse:* $z = x^{-1} \cdot 2^k \bmod p$

SIMD-operations: $[A_1, B_1, A_2, B_2] \leftarrow [A_1 >> t_1, B_1 << t_2, A_2 >> t_2, B_2 << t_1]$

$[A_1, B_1, A_2, B_2] \leftarrow [A_1 - A_2, B_1 - B_2, A_2, B_2]$

$[A_1, B_1, A_2, B_2] \leftarrow [A_1, B_1, A_2 - A_1, B_2 - B_1]$

Branches significantly reduced

Modular Inversion

Algorithm 1 4-SIMD Extended Binary GCD

Input: $p : r^{n-1} < p < r^n$ and $\gcd(p, 2) = 1$
$x : 0 < x < r^n$ and $\gcd(x, p) = 1$

Output: $z \equiv \frac{1}{x} \bmod p$

$[A_1, B_1, A_2, B_2] := [p, 0, x, 1]$ and $[k_1, k_2] := [0, 0]$
while true **do**
  / Start of shift reduction. */*
  Find t_1 such that $2^{t_1} | A_1$
  Find t_2 such that $2^{t_2} | A_2$
  $[k_1, k_2] := [k_1 + t_1, k_2 + t_2]$
  $[A_1, B_1, A_2, B_2] := [A_1 >> t_1, B_1 << t_2, A_2 >> t_2, B_2 << t_1]$

  / Start of subtraction reduction. */*
  if $(A_1 > A_2)$ **then**
    $[A_1, B_1, A_2, B_2] := [A_1 - A_2, B_1 - B_2, A_2, B_2]$
  else if $(A_2 > A_1)$ **then**
    $[A_1, B_1, A_2, B_2] := [A_1, B_1, A_2 - A_1, B_2 - B_1]$
  else
    return $z := B_2 \cdot (2^{-(k_1+k_2)}) \bmod p$
  end if
end while

Performance Results

Operation	#cycles required by each operation	#operation per iteration	#cycles per iteration
Mod Mul	53	6	318
Mod Sub	5	6	30
Partial Mon Red	24	1	24
Mod Inv	4941	1/400	12
Misc.	69	1	69
Total			453

[1 SPU, 4-SIMD @3.2 GHZ]

Hence, our cluster of 214 PS3s computes:
$9.1 \cdot 10^9 \approx 2^{33}$ iterations per sec
It works on $> 0.5\text{M}$ curves in parallel

Performance Comparison

- XC3S1000 FPGAs[1]
  FPGA results of EC over 96 and 128-bit generic prime fields for COPACABANA [2]
  Can host up to 120 FPGAs (Cost: 10’000 USD)

- PlayStation 3 (our implementation)
  Targeted at 112-bit prime curve.
  Use 128-bit multiplication + fast reduction modulo $\tilde{p}$
  For 10’000 USD $\approx$ 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, vol. 4249 of LNCS, pages 101-118, 2006.

Comparison

	96 bits	128 bits
COPACABANA (XC3S1000)	$4.0 \cdot 10^7$	$2.1 \cdot 10^7$
+ 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	$1.4 \cdot 10^9$	

Table: Iterations per second

33 PS3 / COPACABANA (96 bits): 12.4 times faster
33 PS3 / COPACABANA (128 bits): 23.8 times faster
Note: numbers without using 33 dual-threaded PPEs!

The 112-bit Solution

- The point P of order n is given in the standard.
- The x-coordinate of Q was chose 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 run continuously).
- If run continuously, using the latest version of our code, the same calculation would have taken 3.5 months.

$$P = (1882814650\,5797253489\,2223778713\,752, 341987\,5491033170\,8271678618\,96082688)$$

$$Q = (1415926535\,8979323846\,2643383279\,5028, 38467\,5960649470\,6724286139\,623885544)$$

$$n = 4451685225\,0937147764\,9189154254\,8933$$

$$Q = 3125216360\,1477247716\,1767351856\,699 \cdot P$$

Conclusions

- We have measured the hardness of solving the ECDLP on a 112-bit prime field. Requires 62.6 PS3 years to solve it.
- We have presented modular arithmetic algorithms using SIMD instructions. Optimized for 112-bit prime.
- Set a new record for solving the ECDLP.

Do not use the standardized elliptic curve
over 112-bit prime fields!

The PS3 cluster at LACAL

- Cluster room: 190 PS3s
- PlayLaB: 6 x 4 PS3s
  (connected to the cluster)
- Offices: 5 PS3s
  (for programming purposes)
- Total: 219 PS3s