Solving Discrete Logarithms in Smooth-Order Groups with CUDA

WATERLOO
CHERITON SCHOOL OF COMPUTER SCIENCE
cs.uwaterloo.ca

Ryan Henry
Ian Goldberg

Government of Canada
Vanier Canada Graduate Scholarships
Gouvernement du Canada
Bourses d'études supérieures du Canada Vanier
Canada
NSERC CRSNG
Ontario Research Fund
mprime

Definition

Let $\mathbb{G}$ be a cyclic group of order q and let $g \in \mathbb{G}$ be a generator. Given $\alpha \in \mathbb{G}$, the **discrete logarithm (DL) problem** is to find $x \in \mathbb{Z}_q$ such that $g^x = \alpha$.

Definition

Let $\mathbb{G}$ be a cyclic group of order q and let $g \in \mathbb{G}$ be a generator. Given $\alpha \in \mathbb{G}$, the **discrete logarithm (DL) problem** is to find $x \in \mathbb{Z}_q$ such that $g^x = \alpha$.

Why do we care?

- Computing DLs is apparently difficult for classical computers
- Inverse problem (modular exponentiation) is easy
- Many cryptographic protocols exploit this asymmetry

Definition

An integer n is called ***B*-smooth** if each of its prime factors is bounded above by B. A **smooth-order group** is just a group whose order is B-smooth for some “suitably small” value of B.

Definition

An integer n is called ***B*-smooth** if each of its prime factors is bounded above by B. A **smooth-order group** is just a group whose order is B-smooth for some “suitably small” value of B.

Why do we care?

- If $\varphi(N)$ is B-smooth, then $\mathbb{Z}_N^*$ has smooth order
- Many DL-based cryptographic protocols work in $\mathbb{Z}_N^*$
- Pollard’s rho algorithm (plus Pohlig-Hellman) solves DLs in time proportional to smoothness of group order

Definition

The **Compute Unified Device Architecture (CUDA)** is Nvidia's parallel computing architecture. It enables developers to use CUDA-enabled Nvidia GPUs for general purpose computing.

CUDA

Definition

The **Compute Unified Device Architecture (CUDA)** is Nvidia's parallel computing architecture. It enables developers to use CUDA-enabled Nvidia GPUs for general purpose computing.

Why do we care?

- Nvidia GPUs are widely deployed, and offer better price-to-GFLOP ratio than CPUs
- Modern GPUs have many cores and support highly parallel computation
- Pollard's rho algorithm is extremely parallelizable

In this presentation, we...

- **describe Pollard's rho algorithm and its parallel variant**

In this presentation, we...

- describe Pollard's rho algorithm and its parallel variant
- **discuss CUDA and GP GPU computing on Nvidia GPUs**

In this presentation, we...

- describe Pollard's rho algorithm and its parallel variant
- discuss CUDA and GP GPU computing on Nvidia GPUs
- **present our implementation of modular multiplication and parallel rho in CUDA and analyze its performance**

In this presentation, we...

- describe Pollard's rho algorithm and its parallel variant
- discuss CUDA and GP GPU computing on Nvidia GPUs
- present our implementation of modular multiplication and parallel rho in CUDA and analyze its performance
- **point out a simple attack on Boudot's zero-knowledge range proofs**

In this presentation, we...

- describe Pollard's rho algorithm and its parallel variant
- discuss CUDA and GP GPU computing on Nvidia GPUs
- present our implementation of modular multiplication and parallel rho in CUDA and analyze its performance
- point out a simple attack on Boudot's zero-knowledge range proofs
- **construct and analyze trapdoor discrete logarithm groups**

Part I: Pollard’s rho

Pollard's rho algorithm (1/4)

Problem

Given $g, h \in \mathbb{G}$, compute the discrete logarithm $x \in \mathbb{Z}_n$ of h with respect to g.

Pollard's rho algorithm (1/4)

Problem

Given $g, h \in \mathbb{G}$, compute the discrete logarithm $x \in \mathbb{Z}_n$ of h with respect to g.

Key observation:

- Consider elements $g^a h^b \in \mathbb{G}$ and search for collisions
- Since $g^{a_1} h^{b_1} = g^{a_2} h^{b_2} \implies g^{a_1 - a_2} = h^{b_2 - b_1}$, we have $a_1 - a_2 \equiv x\,(b_2 - b_1) \bmod n \Longrightarrow x \equiv (a_1 - a_2)(b_2 - b_1)^{-1} \bmod n$
- **Birthday paradox:** about $\sqrt{\pi\, n/2}$ selections should suffice $\Longrightarrow$ expected runtime and storage in $\Theta(\sqrt{n})$

Pollard's rho algorithm (2/4)

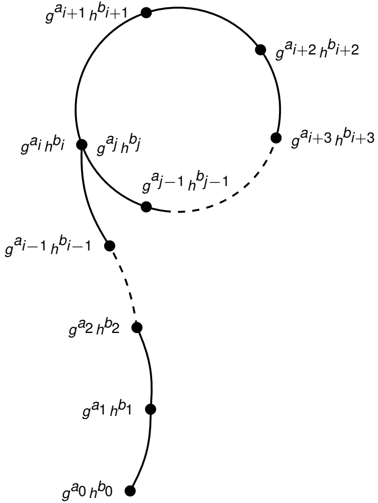
Problem

Given $g, h \in \mathbb{G}$, compute the discrete logarithm $x \in \mathbb{Z}_n$ of h with respect to g.

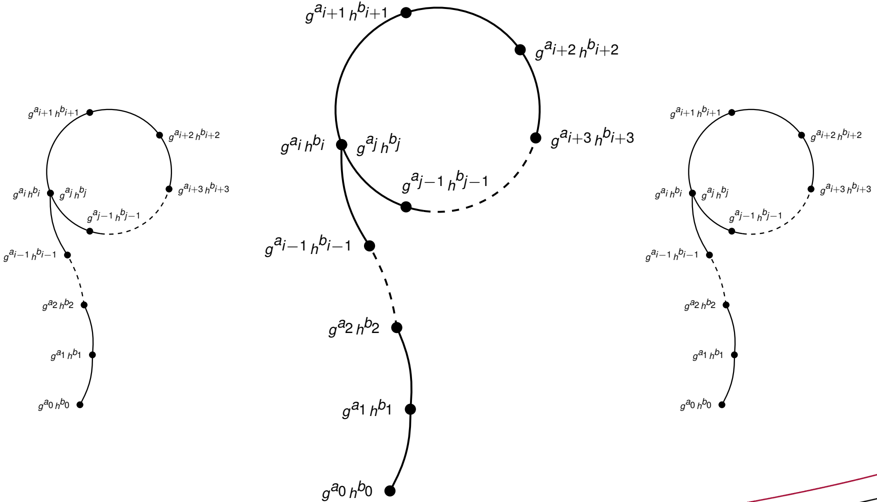
Pollard's idea:

- Walk through $\mathbb{G}$ using **iteration function** $f : \mathbb{G} \to \mathbb{G}$,
$$f(g^{a_i} h^{b_i}) = g^{a_{i+1}} h^{b_{i+1}}$$
- Collisions $\Longrightarrow$ cycles, which are cheap to detect
- If iteration function behaves "randomly enough", then expected runtime is in $\Theta(\sqrt{n})$ and storage is in $\Theta(1)$

Pollard's rho algorithm (3/4)



Pollard's rho algorithm (3/4)



Pollard's rho algorithm (4/4)

Problem

Given $g, h \in \mathbb{G}$, compute the discrete logarithm $x \in \mathbb{Z}_n$ of h with respect to g.

van Oorschot's and Wiener's idea:

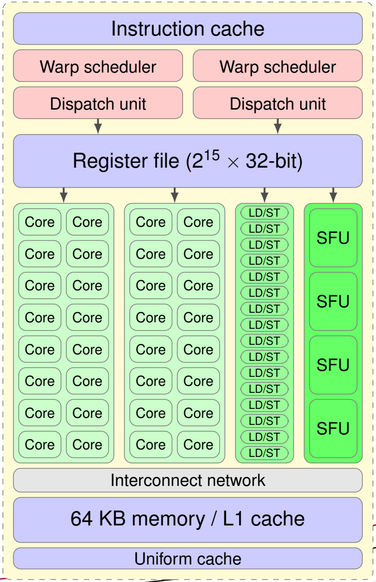
- Define a **distinguished point (DP)** as any point with some cheap-to-detect property (e.g., m trailing zeros)
- Run Ψ client threads in parallel, each reporting DPs to a central server that checks for collisions
- Expected runtime is in $\Theta\left(\sqrt{n}/\Psi\right)$

Part II: GPUs and CUDA

SMPs and CUDA cores

Fermi architecture

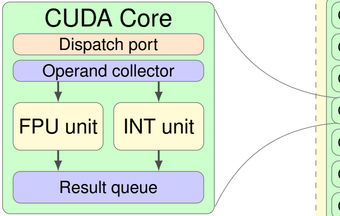
- GPU has several streaming multiprocessors (SMP)
- Our Tesla M2050 cards each have 14 SMPs
- SIMD architecture

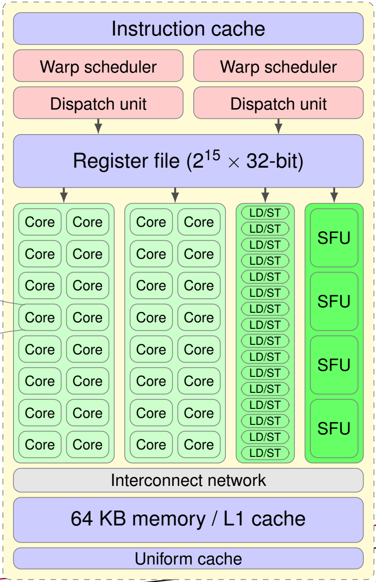


SMPs and CUDA cores

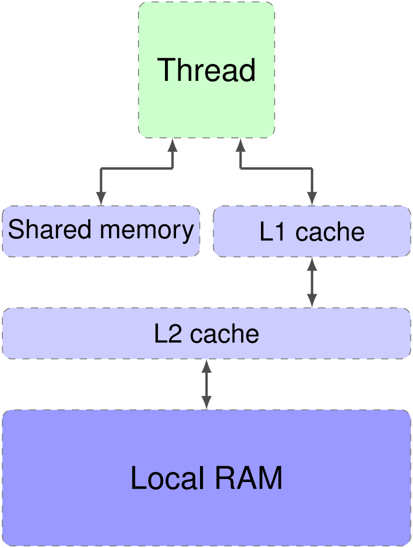
Fermi architecture

- GPU has several streaming multiprocessors (SMP)
- Our Tesla M2050 cards each have 14 SMPs
- SIMD architecture





CUDA memory hierarchy



- Developer manages memory explicitly
- 1 clock pulse for shared memory and L1 cache
- $\approx$ 300 clock pulses for Local RAM
- Many more clock pulses for system RAM

Tesla M2050

Nvidia Tesla M2050 GPU cards:

- Based on Fermi architecture
- 14 SMPs $\times$ 32 $^{\text{cores}}/_{\text{SMP}}$ $=$ 448 cores (each running at 1.55 GHz)
  - 2^{15} $\times$ 32-bit $^{\text{registers}}/_{\text{SMP}}$
  - Configurable: 64 KB shared memory / L1 cache
- 3 GB GDDR5 of Local RAM

amazon.com. price: **1,299.00 USD**

Our experiments used a host PC with:

- Intel Xeon E5620 quad core (2.4 GHz)
- 2 $\times$ 4 GB of DDR3-1333 RAM
- 2$\times$ Tesla M2050 GPU cards

Part III: Implementation

CUDA modular multiplication (1/2)

- Iteration function for Pollard rho:

$$f(x) = \begin{cases} g\,x & \text{if } 0 \le x < \frac{q}{3} \\ x^2 & \text{if } \frac{q}{3} \le x < \frac{2q}{3} \\ h\,x & \text{if } \frac{2q}{3} \le x < q \end{cases}$$

- Need **fast, multiprecision modular multiplication** to solve DLs in Z_N^*
- We used Koç et al's CIOS algorithm for Montgomery multiplication
  - Low auxiliary storage $\implies$ lots of threads
  - We do one thread per multiplication

CUDA modular multiplication (2/2)

Table: k-bit modular multiplications per second and (amortized) time per k-bit modular multiplication ***on a single Tesla M2050.***

Bit length of modulus	Time per trial $\pm$ std dev	Amortized time per modmult	Modmults per second
192	30.538 s $\pm$ 4 ms	1.19 ns	$\approx$ 840,336,000
256	50.916 s $\pm$ 5 ms	1.98 ns	$\approx$ 505,050,000
512	186.969 s $\pm$ 4 ms	7.30 ns	$\approx$ 136,986,000
768	492.6 s $\pm$ 200 ms	19.24 ns	$\approx$ 51,975,000
1024	2304.5 s $\pm$ 300 ms	90.02 ns	$\approx$ 11,108,000

- Larger k $\implies$ each multiplication takes longer
  $\implies$ can compute fewer multiplications in parallel

CUDA Pollard rho (1/2)

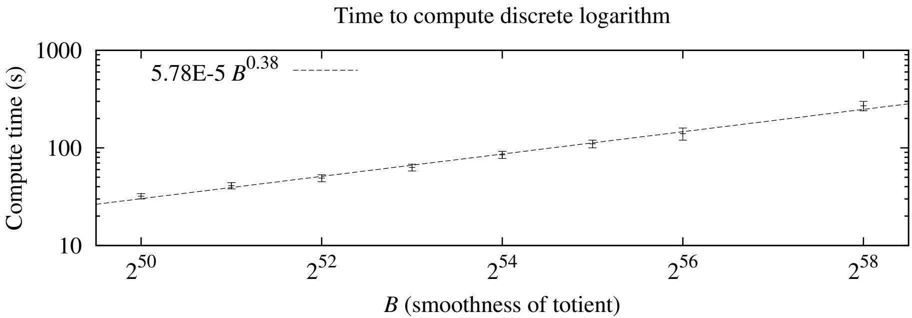
Goal

Compute discrete logarithms modulo k_N-bit RSA numbers $N = pq$ with 2^{k_B}-smooth totient.

Our implementation:

- Optimized for $k_N = 1536$ and $k_B \approx 55$
- Assumes that the factorization of $p - 1$ and $q - 1$ is known
- Uses Pohlig-Hellman approach to decompose problem to k_B-bit subproblems
- Distinguished points: at least 10 trailing zeros in binary (Montgomery) representation

CUDA Pollard rho (2/2)



- Expected cost per B-smooth DL is in $\Theta(\sqrt{B})$
- Each card solves $\frac{768}{\lg B}$ such DLs $\Longrightarrow$ runtime in $\Theta(\sqrt{B}/\lg B)$
- $B \approx 2^{54} \Longrightarrow$ runtime roughly proportional to $B^{0.39}$

Part IV: Implications

Implications

What are the implications for existing DL-based cryptosystems?

In most cases, ***there are no real implications***.

Implications

What are the implications for existing DL-based cryptosystems?

In most cases, ***there are no real implications***.

So why am I speaking at *SHARCS*?

- Cost estimates for cryptographically interesting computations are useful
- Construct trapdoor discrete logarithm groups
- Potential attacks on some zero-knowledge proofs
- Menezes: duplicate signature key selection (DSKS) attacks on RSA

Attack on zero-knowledge "range proofs"

Problem

For a fixed generator $g \in \mathbb{G}$ and commitment $C = g^x$, prove (in zero-knowledge, with knowledge of x) that $a \leq x \leq b$.

Attack on zero-knowledge “range proofs”

Problem

For a fixed generator $g \in \mathbb{G}$ and commitment $C = g^x$, prove (in zero-knowledge, with knowledge of x) that $a \leq x \leq b$.

Lagrange's four square theorem: An integer $x \in \mathbb{Z}$ is nonnegative if and only if it can be expressed as the sum of (at most) four integer squares.

- **Idea**: Compute $C_a = C/g^a = g^{x-a}$ and $C_b = g^b/C = g^{b-x}$, then prove that C_a and C_b each commit to a sum of four squares.

Attack on zero-knowledge "range proofs"

Problem

For a fixed generator $g \in \mathbb{G}$ and commitment $C = g^x$, prove (in zero-knowledge, with knowledge of x) that $a \leq x \leq b$.

Lagrange's four square theorem: An integer $x \in \mathbb{Z}$ is nonnegative if and only if it can be expressed as the sum of (at most) four integer squares.

- **Idea**: Compute $C_a = C/g^a = g^{x-a}$ and $C_b = g^b/C = g^{b-x}$, then prove that C_a and C_b each commit to a sum of four squares.
- **Soundness relies on order of $\mathbb{G}$ being hidden, which it usually is not!**

Attack on zero-knowledge "range proofs"

Problem

For a fixed generator $g \in \mathbb{G}$ and commitment $C = g^x$, prove (in zero-knowledge, with knowledge of x) that $a \leq x \leq b$.

Lagrange's four square theorem: An integer $x \in \mathbb{Z}$ is nonnegative if and only if it can be expressed as the sum of (at most) four integer squares.

- **Idea**: Compute $C_a = C/g^a = g^{x-a}$ and $C_b = g^b/C = g^{b-x}$, then prove that C_a and C_b each commit to a sum of four squares.
- **Soundness relies on order of $\mathbb{G}$ being hidden, which it usually is not!**
- Move proof into $\mathbb{Z}_N^*$ for RSA number $N = pq$ (whose factorization is kept secret from the prover)

Trapdoor discrete logarithm groups (1/3)

Idea

Work modulo an RSA modulus $N = pq$ such that $p - 1$ and $q - 1$ are B-smooth.

- **Public key:** N
- **Private key:** p, q and the factorization of $p - 1$ and $q - 1$

Trapdoor discrete logarithm groups (1/3)

Idea

Work modulo an RSA modulus $N = pq$ such that $p - 1$ and $q - 1$ are B-smooth.

- **Public key:** N
- **Private key:** p, q and the factorization of $p - 1$ and $q - 1$

Trapdoor DL cost

- **With trapdoor key:** DL computation takes $\Theta\left(\frac{\lg N}{\lg B}\sqrt{B}\right)$ highly parallelizable work
- Let μ_1 be the number of $(\lg N/2)$-bit modular multiplications computable per core-second, then trapdoor DL runtime is

$$\approx \frac{\lg N}{\lg B} \cdot \frac{c \cdot \sqrt{B}}{\Psi \cdot \mu_1} \text{ seconds,}$$

for some constant c.

Trapdoor discrete logarithm groups (2/3)

Idea

Work modulo an RSA modulus $N = pq$ such that $p - 1$ and $q - 1$ are B-smooth.

- **Public key:** N
- **Private key:** p, q and the factorization of $p - 1$ and $q - 1$

Non-trapdoor DL cost (1/2)

- **Without trapdoor key:** Best approach seems to be factoring to recover private key!
- **Pollard's $p - 1$ algorithm:** Factors B-smooth numbers with $O(B)$ work
- **$p - 1$ attack is inherently serial! Parallelism won't help much!**

Trapdoor discrete logarithm groups (2/3)

Idea

Work modulo an RSA modulus $N = pq$ such that $p - 1$ and $q - 1$ are B-smooth.

- **Public key:** N
- **Private key:** p, q and the factorization of $p - 1$ and $q - 1$

Non-trapdoor DL cost (2/2)

- **ECM, QS, et al.**: highly parallelizable and subexponential cost, **but cost scales with lg N instead of B**
- For 1536-bit RSA moduli, cross over point occurs when $\Psi \cdot B \approx 2^{85}$
- Need $\Psi \gg 2^{30}$ cores to do faster non-trapdoor DL with other algorithms

Trapdoor discrete logarithm groups (3/3)

Idea

Work modulo an RSA modulus $N = pq$ such that $p - 1$ and $q - 1$ are B-smooth.

- **Public key:** N
- **Private key:** p, q and the factorization of $p - 1$ and $q - 1$

Practical security analysis

$$B \approx 2^{55} \Longrightarrow \begin{cases} > 1700 \text{ years} & \text{for non-trapdoor DL} \\ < 2 \text{ minutes} & \text{for trapdoor DL} \end{cases}$$

- These are ***wall-clock*** times!

Part V: Conclusion

Summary

- Used CUDA to solve DLs in smooth-order groups
- Up to about 2^{58}-smooth 1536-bit RSA numbers in under 5 minutes on $2 \times$ Tesla M2050
  - $>$ 100 million 768-bit modular multiplications per second
  - $>$ 1.7 billion 192-bit modular multiplications per second
  - **Extrapolating**: 2^{80}-smooth DL should be feasible in $\approx$ 23 hours on same Tesla cards (with a bit more system RAM)
- Constructed and analyzed trapdoor discrete logarithm groups
- Proposed simple attack on (naively implementations of) Boudot's zero-knowledge range proofs

All of our code is free and open source:

http://crysp.uwaterloo.ca/software/