Solving Discrete Logarithms in Smooth-Order Groups with CUDA

Ryan Henry
Ian Goldberg

WATERLOO
CHERITON SCHOOL OF
COMPUTER SCIENCE

Ryan Henry
Ian Goldberg
Definition

Let $G$ be a cyclic group of order $q$ and let $g \in G$ be a generator. Given $\alpha \in G$, 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

Let $G$ be a cyclic group of order $q$ and let $g \in G$ be a generator. Given $\alpha \in G$, 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 **Computed Unified Device Architecture (CUDA)** is Nvidia’s parallel computing architecture. It enables developers to use CUDA-enabled Nvidia GPUs for general purpose computing.
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 \( a_1-a_2 \equiv x (b_2-b_1) \mod n \implies x \equiv (a_1-a_2)(b_2-b_1)^{-1} \mod n \)
- Birthday paradox: about \( \sqrt{\pi n/2} \) selections should suffice \( \implies \) expected runtime and storage in \( \Theta(\sqrt{n}) \)
Pollard’s rho algorithm (2/4)

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

Pollard’s idea:
- Walk through \( G \) using {\textbf{iteration function}} \( f : G \rightarrow G \),
  \[
  f(g^{a_i} h^{b_i}) = g^{a_{i+1}} h^{b_{i+1}}
  \]
- Collisions \( \Rightarrow \) 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)
Problem
Given $g, h \in G$, 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 $\Psi$ client threads in parallel, each reporting DPs to a central server that checks for collisions
- Expected runtime is in $\Theta(\sqrt{n}/\Psi)$
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
- ≈ 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/SMP} = 448 \text{ cores} (each running at 1.55 \text{ GHz})
  - \(2^{15} \times 32\)-bit registers/SMP
  - Configurable: 64 KB shared memory / L1 cache
- 3 GB GDDR5 of Local RAM

Our experiments used a host PC with:

- Intel Xeon E5620 quad core (2.4 \text{ GHz})
- 2 \times 4 \text{ GB of DDR3-1333 RAM}
- 2 \times \text{Tesla M2050 GPU cards}
Part III: Implementation
CUDA modular multiplication (1/2)

- Iteration function for Pollard rho:
  \[
  f(x) = \begin{cases} 
  g \cdot x & \text{if } 0 \leq x < \frac{q}{3} \\
  x^2 & \text{if } \frac{q}{3} \leq x < \frac{2q}{3} \\
  h \cdot x & \text{if } \frac{2q}{3} \leq x < q
  \end{cases}
  \]

- Need fast, multiprecision modular multiplication to solve DLs in \( \mathbb{Z}_N^* \)

- We used Koç et al’s CIOS algorithm for Montgomery multiplication
  - Low auxiliary storage \(\Rightarrow\) 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.

<table>
<thead>
<tr>
<th>Bit length of modulus</th>
<th>Time per trial ± std dev</th>
<th>Amortized time per modmult</th>
<th>Modmults per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>192</td>
<td>30.538 s ± 4 ms</td>
<td>1.19 ns</td>
<td>≈ 840,336,000</td>
</tr>
<tr>
<td>256</td>
<td>50.916 s ± 5 ms</td>
<td>1.98 ns</td>
<td>≈ 505,050,000</td>
</tr>
<tr>
<td>512</td>
<td>186.969 s ± 4 ms</td>
<td>7.30 ns</td>
<td>≈ 136,986,000</td>
</tr>
<tr>
<td>768</td>
<td>492.6 s ± 200 ms</td>
<td>19.24 ns</td>
<td>≈ 51,975,000</td>
</tr>
<tr>
<td>1024</td>
<td>2304.5 s ± 300 ms</td>
<td>90.02 ns</td>
<td>≈ 11,108,000</td>
</tr>
</tbody>
</table>

- 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 $\implies$ runtime in $\Theta(\sqrt{B}/\lg B)$

$B \approx 2^{54} \implies$ 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 \( a \leq x \leq b \).
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.
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!**
**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 \( \mu_1 \) be the number of \((\lg N/2)\)-bit modular multiplications computable per core-second, then trapdoor DL runtime is

\[
\approx \frac{\lg N}{\lg B} \cdot \frac{c \cdot \sqrt{B}}{\psi \cdot \mu_1}
\]

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

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

- 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/