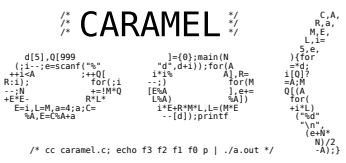
# RNS Arithmetic for Linear Algebra of Discrete Logarithm Computations Using Parallel Architectures

Hamza Jeljeli

CARAMEL project-team, LORIA, INRIA / CNRS / Université de Lorraine,
Hamza.Jeljeli@loria.fr

RAIM 2015, Rennes, April 8th, 2015







# Discrete Logarithm Problem (DLP)

## Discrete Logarithm

Given a cyclic group $G = \langle g \rangle$ written multiplicatively, the discrete logarithm of $h \in G$ is the unique $k$ in $[0, \#G - 1]$ s.t. $h = g^k$.

- In some groups, DLP is computationally hard.
- The inverse problem (discrete exponentiation) is easy.
- Security of **cryptographic primitives** relies on difficulty of DLP:
  - key agreement: Diffie–Hellman key exchange,
  - encryption: ElGamal encryption,
  - signature: DSA signature,
  - pairing-based cryptography, ...


- Evaluate security level of primitives $\Longrightarrow$ DLP attacks.

# Linear Algebra Issued from DLP Attacks

- Focus on DLP in **multiplicative subgroups of finite fields GF($q$)**.
- To attack DLP in finite fields, *index-calculus* methods:
  - solve DLP in time sub-exponential or quasi-polynomial in the size of the finite field;
  - require solving large sparse systems of linear equations over finite fields.

## Linear Algebra Problem

**Inputs:** a prime $\ell$ that divides $q-1$ and a matrix $A$.
**Output:** a non-trivial vector $w$ s.t. $Aw \bmod \ell = 0$.

**Linear Algebra for Factorization**

- Arithmetic over GF(2).
- 10% of overall time.

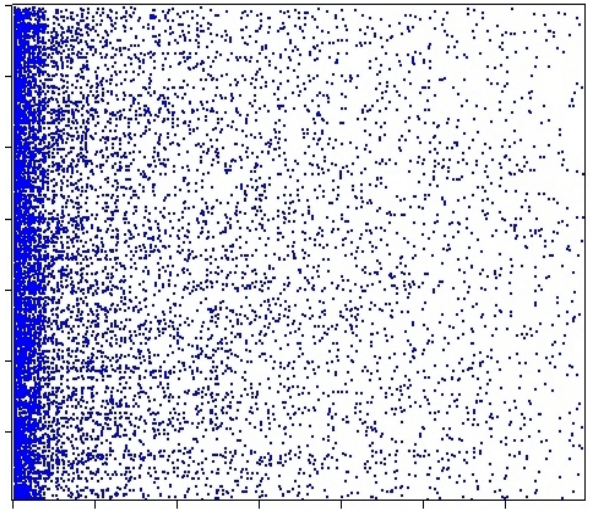
**Linear Algebra for DLP**

- Arithmetic over GF($\ell$).
- 50% of overall time.
- Bottleneck for computation.

# Characteristics of the Inputs

- $\ell$ between 100 and 1000 bits.
- $A$ is an $N$-by-$N$ matrix, $N$ ranges from $10^5$ to $10^8$.
- $A$ is sparse, each row of $A$ contains $\sim 100$ of non-zero coefficients.

- The very first columns are relatively dense, then the column density decreases gradually.
- The row density does not change significantly.
- Non-zero coefficients in GF($\ell$).



**Example:** Resolution of DLP in GF($2^{619}$)$^\times$

| | |
|---|---|
| Size of $\ell$ | 217 bits |
| Size of matrix ($N$) | 650k |
| Average row weight | 100 |

# Linear Algebra

- Harder linear algebra $\Longrightarrow$ heavy computations, exploit parallelism:
  1. Algorithmic level: Sparse linear algebra algorithms
     - **Wiedemann**: Sequence of $O(N)$ iterative Sparse-Matrix–Vector products
       $$^txy, {}^txAy, {}^txA^2y, \ldots, {}^txA^{2N}y$$
     - **Block Wiedemann**: Distribute in many parallel sequences

     } Euro-Par 2014
  2. SpMV level:
     - parallelize SpMV over many nodes.
  3. Per-node level:
     - Hardware: GPU, multi-core CPU, many-core, ...?
     - Format for sparse matrix?
     - How to *map* partial SpMV on the architecture?

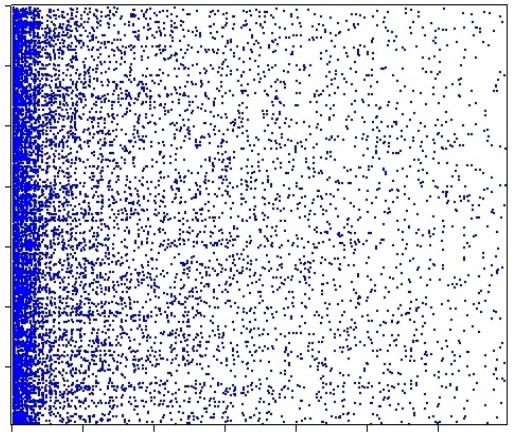
     } WAIFI 2014
  4. Arithmetic level: arithmetic over GF($\ell$).
     - Representation: Residue Number System (RNS), Multi-precision?
     - Accelerate arithmetic over SIMD architectures.

# Table of Contents

# Nature of the Coefficients of $A$

## FFS-like matrices

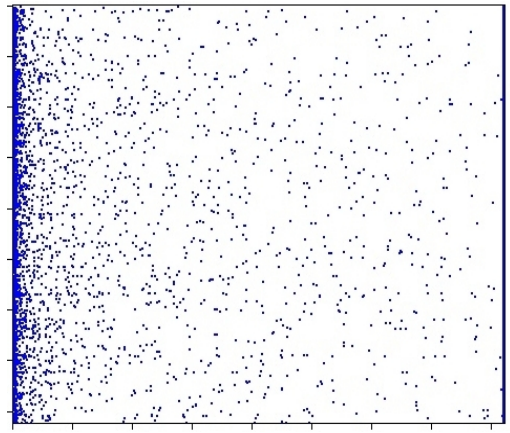
- $A$ is sparse.
- All coefficients are "small" ($|.| \in [0, 2^{10}]$).
- $\sim 90\%$ are $\pm 1$.



## NFS-like matrices

Composed of 2 parts:

- $A_0$: a sparse $N$-by-$(N - r)$ sub-matrix containing "small" coefficients (majority of $\pm 1$).
- $A_1$: a dense $N$-by-$r$ sub-matrix composed of "large" ($\in [0, \ell]$) coefficients.
- $r$ is between 0 and 10.



# Required Operations for SpMV

SpMV level: $v \leftarrow Au \bmod \ell$

$\wr$

Row $i$ level:

## FFS-like matrices

$$v_i \leftarrow \sum_{j=1}^{N} a_{ij} u_j \bmod \ell$$

$\wr$

- $v_i \leftarrow v_i \pm u_j, \quad (a_{ij} = \pm 1)$ frequent
- $v_i \leftarrow v_i + a_{ij} \times u_j, \ (|a_{ij}| < 2^{10})$ less frequent
- $v_i \leftarrow v_i \bmod \ell$ (lazy reduction) not frequent

## NFS-like matrices

$$v_i \leftarrow \sum_{j=1}^{N-r} a_{ij} u_j + \sum_{j=N-r+1}^{N} a_{ij} u_j \bmod \ell$$

$\wr$

- $v_i \leftarrow v_i \pm u_j, \quad (a_{ij} = \pm 1)$ frequent
- $v_i \leftarrow v_i + a_{ij} \times u_j, \ (|a_{ij}| < 2^{10})$ less frequent
- $v_i \leftarrow v_i + a_{ij} \times u_j, \ (0 \leq a_{ij} < \ell)$ less frequent
- $v_i \leftarrow v_i \bmod \ell$ (lazy reduction) not frequent

# Table of Contents

# A Brief Reminder on Residue Number System (RNS)

- RNS basis: set of $n$ co-prime integers $(p_1, \ldots, p_n)$, $P = \prod_{i=1}^{n} p_i$.
- RNS representation of $x \in [0, P-1]$: $\vec{x} = (|x|_{p_1}, \ldots, |x|_{p_n})$.
- Usual operations in RNS:
  - Addition: $\overrightarrow{x+y} = (|x_1 + y_1|_{p_1}, \ldots, |x_n + y_n|_{p_n})$
  - Multiplication by scalar $\lambda < p_i$: $\overrightarrow{x \times \lambda} = (|x_1 \times \lambda|_{p_1}, \ldots, |x_n \times \lambda|_{p_n})$
  - Multiplication: $\overrightarrow{x \times y} = (|x_1 \times y_1|_{p_1}, \ldots, |x_n \times y_n|_{p_n})$
  - Operations are $\bmod P$ (final result should not exceed $P$ ⚠).

  $\Rightarrow$ Fully independent parallel computations on the components.
  - Comparison, Division in RNS are more tricky.
- $p_i$ chosen of pseudo-Mersenne form $2^k - c_i$ to speed up $|.|_{p_i}$:
  - $2^k$ a power of a machine word : $2^{32}$, $2^{64}$, ...
  - $c_i$ small compared to $2^k$.

# RNS Addition and Multiplication - Algorithms

- $x + y$ needs that $2 \times (\ell - 1) < P - 1$:

---

**Input** : $\vec{x}$, $\vec{y}$: RNS representations of $x$, $y \in \mathbb{Z}/\ell\mathbb{Z}$.
**Output**: $\vec{z}$: RNS representation of $z = x + y$

**for each** *component* $i$ **do**
| $z_i \leftarrow |x_i + y_i|_{p_i}$

---

- $x + \lambda \times y$, with $\lambda < 2^{10}$, needs that $2^{10} \times (\ell - 1) < P - 1$:

---

**Input** : $\vec{x}$, $\vec{y}$: RNS representations of $x$, $y \in \mathbb{Z}/\ell\mathbb{Z}$ and $\lambda \in [2, 2^{10}[$.
**Output**: $\vec{z}$ : RNS representation of $z = x + \lambda \times y$

**for each** *component* $i$ **do**
| $z_i \leftarrow |x_i + \lambda \times y_i|_{p_i}$

---

- $x + \lambda \times y$, with $\lambda < \ell$, needs that $\ell \times (\ell - 1) < P - 1$:

---

**Input** : $\vec{x}$, $\vec{y}$, $\vec{\lambda}$: RNS representations of $x$, $y$, $\lambda \in \mathbb{Z}/\ell\mathbb{Z}$
**Output**: $\vec{z}$ : RNS representation of $z = x + \lambda \times y$

**for each** *component* $i$ **do**
| $z_i \leftarrow |x_i + \lambda_i \times y_i|_{p_i}$

---

# RNS Reduction Modulo $\ell$ [Bernstein 94]

Problem: We have an $x$ in RNS, $x \bmod \ell$ ?

- Chinese Remainder Theorem (CRT) reconstruction:

$$x = \sum_{i=1}^{n} \gamma_i \cdot P_i \bmod P, \text{ where } P_i = \frac{P}{p_i},\ \gamma_i \triangleq \left| x_i \cdot |P_i^{-1}|_{p_i} \right|_{p_i}$$

$$x = \sum_{i=1}^{n} \gamma_i P_i - \alpha P, \text{ where } \alpha \triangleq \left\lfloor \frac{\sum_{i=1}^{n} \gamma_i P_i}{P} \right\rfloor = \left\lfloor \sum_{i=1}^{n} \frac{\gamma_i}{p_i} \right\rfloor$$

- If $\alpha$ known $\Rightarrow z \triangleq \sum_{i=1}^{n} \gamma_i \, |P_i|_\ell - |\alpha P|_\ell$
- $z$ satisfies $\begin{cases} z & \equiv & x \pmod{\ell} \\ z & \in & [0, \ell \sum_{i=1}^{n} p_i [ \end{cases}$

$\Rightarrow$ Full RNS computation of $z$.

$\Rightarrow$ $z$ is not exact reduction of $x$. However, approximate reduction guarantees that intermediate results of SpMV computation do not exceed a bound that we impose less than $P$.

# RNS Approximate Reduction Modulo $\ell$ - Algorithm

**Pre calculation**: Vector $(\left|{P_i}^{-1}\right|_{p_i})$ for $i \in \{1, \ldots, n\}$

Table of RNS representations $\overrightarrow{|P_j|_\ell}$ for $j \in \{1, \ldots, n\}$

Table of RNS representations $\overrightarrow{|\alpha P|_\ell}$ for $\alpha \in \{1, \ldots, n-1\}$

**Input** : $\vec{x}$ : RNS representation of $x$, with $0 \leq x < P$

**Output** : $\vec{z}$: RNS representation of $z \equiv x \pmod{\ell}$, with $z < \ell \sum_{i=1}^{n} p_i$

**for each** *component* $i$ **do**

$\gamma_i \leftarrow \left| x_i \times \left|{P_i}^{-1}\right|_{p_i} \right|_{p_i}$ /* 1 RNS product */

broadcast $\gamma_i$

compute $\alpha$ /* addition of $n$ $s$-bit terms */

**for each** *component* $i$ **do**

$z_i \leftarrow \left| \sum_{j=1}^{n} \gamma_j \times \left| |P_j|_\ell \right|_{p_i} \right|_{p_i}$ /* $(n-1)$ RNS additions & $n$ RNS products */

$z_i \leftarrow \left| z_i - \left| |\alpha P|_\ell \right|_{p_i} \right|_{p_i}$ /* 1 RNS subtraction */

# Required Operations for SpMV in RNS

SpMV level: $v \leftarrow Au \bmod \ell$

$\wr$

Row $i$ level:

## FFS-like matrices

$$v_i \leftarrow \sum_{j=1}^{N} a_{ij} u_j \bmod \ell$$

$\wr$

- $v_i \leftarrow v_i \pm u_j, \quad (a_{ij} = \pm 1)$
  frequent and easy
- $v_i \leftarrow v_i + a_{ij} \times u_j, \ (|a_{ij}| < 2^{10})$
  less frequent and easy
- $v_i \leftarrow v_i \bmod \ell$ (lazy reduction)
  not frequent and hard

## NFS-like matrices

$$v_i \leftarrow \sum_{j=1}^{N-r} a_{ij} u_j + \sum_{j=N-r+1}^{N} a_{ij} u_j \bmod \ell$$

$\wr$

- $v_i \leftarrow v_i \pm u_j, \quad (a_{ij} = \pm 1)$
  frequent and easy
- $v_i \leftarrow v_i + a_{ij} \times u_j, \ (|a_{ij}| < 2^{10})$
  less frequent and easy
- $v_i \leftarrow v_i + a_{ij} \times u_j, \ (0 \leq a_{ij} < \ell)$
  less frequent and easy but binding
- $v_i \leftarrow v_i \bmod \ell$ (lazy reduction)
  not frequent and hard

# How long is the RNS Basis?

**FFS-like matrices:**

1. Take a basis $\mathcal{B}(n,k)$ that handles the product by $A$.

Let $s$ be the maximal norm of the rows of $A$:

$$s\ell \sum_{i=1}^{n} p_i < P \text{ (Recall that Wiedemann is iterative).}$$

**NFS-like matrices:**

1. Take a minimal-length basis $\mathcal{B}(n,k)$ when multiplying by $A_0$
2. Extend to a larger basis $\mathcal{B}||\hat{\mathcal{B}}(n+\hat{n},k)$ when multiplying by $A_1$

$$\begin{cases} s\ell(\sum_{i=1}^{n} p_i + \sum_{i=1}^{\hat{n}} \hat{p}_i) & < & P & \text{(product by } A_0) \\ r\ell \times s\ell(\sum_{i=1}^{n} p_i + \sum_{i=1}^{\hat{n}} \hat{p}_i) & < & P\hat{P} & \text{(product by } A_1). \end{cases}$$

## Basis extension: approach similar to reduction modulo $\ell$

For each modulus $\hat{p}_j$ of the new basis:

$$\hat{x}_j = |x|_{\hat{p}_j} = \left| \sum_{i=1}^{n} \gamma_i \, |P_i|_{\hat{p}_j} - |\alpha P|_{\hat{p}_j} \right|_{\hat{p}_j} .$$

# RNS over Parallel Architectures

**GPUs:**

- Multi-threaded architecture: large number of threads running in parallel according to SPMD (Single Program Multiple Data) model.

**SIMD extensions for CPUs:**

- Vectorization: a single instruction performs in parallel an operation on multiple data.

  **Example:** AVX2 extension
  256-bit register packs (integer or floating point) components:

  ➢ 4 64-bit components,
  ➢ 8 32-bit components, ...

**Assign RNS components over GPUs/SIMD**

Operations on $n$ RNS components performed in parallel by $n$ threads/vectorial units.

# Table of Contents

# RNS Arithmetic Implementation

## **GPUs:** inline assembly (PTX)

```
#define __modadd_gpu( c_hi, c_lo, a_hi, a_lo, b_hi, b_lo, pc ) \
        asm( "{\n\t"                                    \
             ".reg .s32 t;"                             \
             "add.cc.u32 %1, %3, %5;"                   \
             "addc.cc.u32 %0, %2, %4;"                  \
             "addc.s32 t, -1, 0;"                       \
             "slct.u32.s32 t, %6, 0, t;"                \
             "add.cc.u32 %1, %1, t;"                    \
             "addc.u32 %0, %0, 0;"                      \
             "}"                                        \
             : "=r" (c_hi), "=r" (c_lo)                 \
             : "r" (a_hi), "r" (a_lo), "r" (b_hi), "r" (b_lo), "r" (pc) )
```

## **SIMD extensions for CPUs:** intrinsics

```
static inline __m256i modadd_256 (__m256i a, __m256i b, __m256i p)
{
    __m256i temp, res;
    res = _mm256_add_epi64 (a, b);
    temp = _mm256_cmpgt_epi64 (_mm256_setzero_si256(), res);
    temp = _mm256_and_si256 (p, temp);
    return _mm256_sub_epi64 (res, temp);
}
```

# RNS Arithmetic Performance

**Matrix:** matrix from DLP in $\mathrm{GF}(2^{619})^{\times}$ using FFS algorithm

| | |
|---|---|
| Size of $\ell$ | 217 bits |
| Size of matrix ($N$) | 650k |
| Average row weight | 100 |

**GPUs:** NVIDIA GeForce GTX 680 (Kepler)

| Operation | $x \pm y$ | $x \pm \lambda y$ | $x \bmod \ell$ | SpMV |
|---|---|---|---|---|
| Occurrence ratio | 91.7% | 7.2% | 1% | 100% |
| Time with PTX | **102 cycles** | **324 cycles** | **5216 cycles** | **27.1 ms** |

**SIMD extensions for CPUs:** 1 core of Intel i5-4570 (3.2 GHz)

| | | | | |
|---|---|---|---|---|
| Time with MMX | 17.3 cycles | 83.1 cycles | 1779 cycles | 1561 ms |
| Time with SSE2 | 11.1 cycles | 51.3 cycles | 1183 cycles | 1007 ms |
| Time with AVX2 | **7.9 cycles** | **26.9 cycles** | **643 cycles** | **598 ms** |

# Comparison of RNS and Multi-precision Arithmetics

**Matrix:** matrix from DLP in GF$(2^{619})^\times$ using FFS algorithm

| | |
|---|---|
| Size of $\ell$ | 217 bits |
| Size of matrix ($N$) | 650k |
| Average row weight | 100 |

**GPUs:** NVIDIA GeForce GTX 680 (Kepler)

| Operation | $x \pm y$ | $x \pm \lambda y$ | $x \bmod \ell$ | SpMV |
|---|---|---|---|---|
| Occurrence ratio | 91.7% | 7.2% | 1% | 100% |
| Time with MP | 184 cycles | **281 cycles** | **361 cycles** | 31 ms |
| Time with RNS | **102 cycles** | 324 cycles | 5216 cycles | **27.1 ms** |

$\Rightarrow$ Speed-up of RNS compared to MP on SpMV time around $15\%$.

**SIMD extensions for CPUs:** 1 core of Intel i5-4570 (3.2 GHz)

| | | | | |
|---|---|---|---|---|
| Time with MP (GMP `mpn`) | 15.2 cycles | **26.3 cycles** | **27.9 cycles** | 782 ms |
| Time with RNS | **7.9 cycles** | 26.9 cycles | 1183 cycles | **598 ms** |

$\Rightarrow$ Speed-up of RNS compared to MP on SpMV time around $30\%$.

# DLP Records

| DLP Comput. | Algo. | $N$ | Size of $\ell$ | Setup | Linear Algebra Wall-clock Time |
|---|---|---|---|---|---|
| GF($2^{619}$) | FFS | 650 k | 217 bits | 1 GPU (GTX 580) | 17 h |
| GF($2^{809}$) | FFS | 3.6 M | 202 bits | 8 GPUs (Tesla M2050) | 4.5 d |
| GF($p_{180}$) | NFS | 7.3 M | 595 bits | Cluster of 768 CPU cores | 39 d |

# Thank you for your attention!


H. Jeljeli. Accelerating Iterative SpMV for Discrete Logarithm Problem Using GPUs, *WAIFI 2014*, pages 25-44, 2015.

H. Jeljeli. Resolution of Linear Algebra for the Discrete Logarithm Problem using GPU and Multi-core Architectures, *Euro-Par 2014 Parallel Processing*, pages 764-775, 2014.

R. Barbulescu, C. Bouvier, J. Detrey, P. Gaudry, H. Jeljeli, E. Thomé, M. Videau, et P. Zimmermann. Discrete logarithm in GF($2^{809}$) with FFS, *PKC 2014*, pages 221-238, 2014.