

---

# Computing discrete logs in large characteristic 2 finite fields

Emmanuel Thomé

`Emmanuel.Thome@polytechnique.fr`

Laboratoire d'Informatique (LIX)  
École polytechnique  
France



1. Introduction – Finite field discrete logs
2. Subexponential algorithms for  $\mathbb{F}_{2^n}$
3. Different sieve techniques
4. Linear algebra
5. Using and improving the Block Wiedemann algorithm
6. Runtime figures:  $\mathbb{F}_{2^{607}}$
7. Conclusion

# 1. Introduction

---

**Discrete logarithms** (DL) appear in several cryptographical settings:

- Diffie-Hellman key exchange.
- El Gamal cryptosystem, El Gamal & Schnorr signature scheme.
- Most ID-based cryptosystems are based on DL instances.

Different groups have been proposed for implementation:

- Finite fields.
- (hyper-)Elliptic curves . . .
- XTR.

# Finite field DL : What for ?

---

- Attack cryptosystems directly based on finite field DL.
- With the Tate pairing, for  $E$  elliptic curve over  $\mathbb{F}_q$ , DL on  $E$  reduces to DL on  $\mathbb{F}_{q^k}$ , with  $k$  s.t.  $\#E(\mathbb{F}_q) \mid q^k - 1$ .  
(MOV reduction on supersingular elliptic curves).

Accounts on the (in)feasibility of DL computations in given finite fields are of high interest to **many** cryptosystems, including recent ID-based schemes.

Subexponential algorithms for this task:

- $\mathbb{F}_p$ : Number field sieve (Gordon, Schirokauer)
- $\mathbb{F}_{2^n}$ : Function field sieve (Adleman), Coppersmith

Common heuristic complexity:

$$L[1/3] = O(\exp((c + o(1))(\log \#K)^{1/3}(\log \log \#K)^{2/3})).$$

Which field sizes can be reached today ?

We focus on characteristic 2 fields,  
trying to improve on Coppersmith's algorithm.

Record presented:  $\mathbb{F}_{2^{607}}$

Another recent computation:  $\mathbb{F}_{2^{521}}$  by Joux and Lercier, using FFS  
(Function field sieve).

## 2. Coppersmith's algorithm for $\mathbb{F}_{2^n}$

$\mathbb{F}_{2^n} = \mathbb{F}_2[X]/(f(X))$ , where  $f(X) = X^n + f_1(X)$  irreducible,  
 $\deg f_1$  small.

Index-calculus type algorithm:

- Choose a **factor base**:  $\mathcal{B} = \{\pi_i \in \mathbb{F}_2[X], \pi_i \text{ irreducible, } \deg \pi_i \leq b\}$ .
- Gather sufficiently many  $(\frac{2^{b+1}}{b})$  relations between the  $\pi_i$ 's.
- Obtain the  $\log \pi_i$  as the solution of a **linear system**.
- $\mathcal{B}$  is chosen large enough so that it is then easy to express the log of anything as a linear combination of the  $\log \pi_i$ .

# Building relations in Coppersmith's algorithm

Consider the **pairs**  $(A, B)$  of polynomials of degree  $\leq d$  (here,  $d = 28$ ).  
Define:

$$C = AX^h + B,$$

$$D = A^k X^{hk-n} f_1 + B^k, \text{ where } n = 607, k = 4, h = 152.$$

One easily verifies that  $C^k \equiv D \pmod{f(X)}$ .  $\deg C \leq 180$ ,  $\deg D \leq 112$ .  
If both  $C$  and  $D$  are  **$b$ -smooth** (all their factors have degree  $\leq b$ ):

$$\prod_i \pi_i^{ke_i} \equiv \prod_i \pi_i^{f_i} \pmod{f(X)},$$

$$\sum_i (f_i - ke_i) \log \pi_i \equiv 0 \pmod{2^n - 1}.$$

Choosing parameters  $b, d$ , is far from easy. Here we took  $b = 23$ , hence  $\#\mathcal{B} = 766, 150$ .

# Parallel with the Function Field Sieve

Coppersmith's algorithm is a special case of the Function Field Sieve.

Let  $H$  be the curve over  $\mathbb{F}_2$  defined by  $H(X, Y) = Y^k + X^{hk-n} f_1(X)$ .

Let  $F$  be the field of rational functions on  $H$ :  $F = \mathbb{F}_2(X, Y)/H(X, Y)$ .

$$\Pi : \begin{cases} F & \rightarrow \mathbb{F}_{2^n} = \mathbb{F}_2[X]/f(X), \\ Y & \mapsto X^h. \end{cases}$$

Consider a function on  $H$ :  $\phi = A(X)Y + B(X) \in F$ . We are interested in:

- Smoothness of  $\Pi(\phi) = A(X)X^h + B(X)$  as a polynomial in  $X$ .
- Smoothness of  $\phi$  in  $F$  (as a sum of principal divisors). Its norm is:

$$A(X)^k H\left(X, \frac{B(X)}{A(X)}\right) = A(X)^k X^{hk-n} f_1(X) + B(X)^k.$$

In this context, the choice of  $H(X, Y)$  can be refined.

---

In short: for a good  $(A, B)$  pair,  $C$  and  $D$  are smooth  $\Rightarrow$  one relation.

**We concentrate on:**

How to efficiently obtain relations (find good  $(A, B)$  pairs) ?

How to solve the resulting linear system ?

### 3. Building relations: different sieving techniques

---

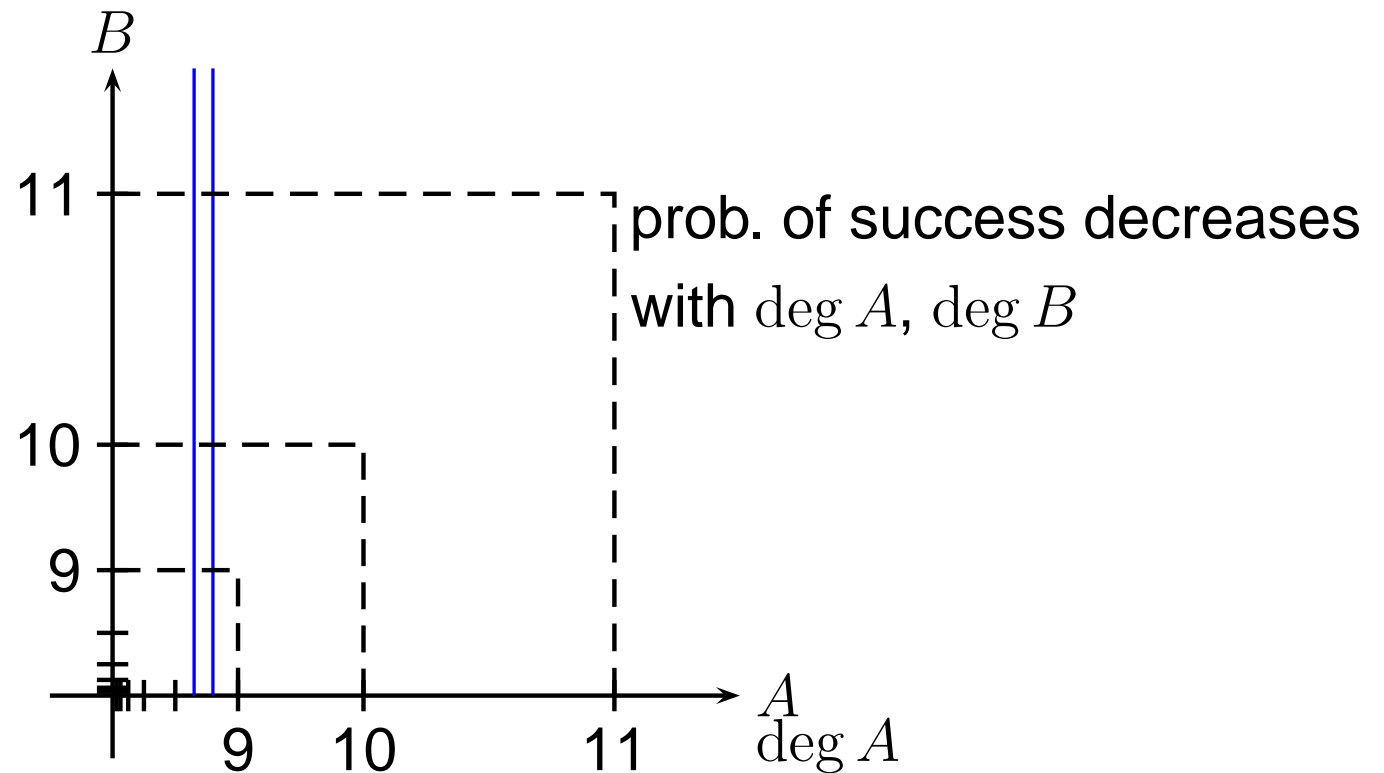
It is good to **sieve** in order to find appropriate  $(A, B)$  pairs: restrict our scope to the pairs where  $C$  has a good chance of being smooth.

First method: **fix**  $A$ , and try to spot  $B$ 's such that  $(A, B)$  yields a smooth  $C$ . For each irreducible  $g$  in the factor base, consider:

$$\begin{aligned} \mathcal{S}(g) &= \{B \text{ s.t. } C = AX^h + B \equiv 0 \pmod{g}\}, \\ &= \text{affine subspace of } \langle 1, X, \dots, X^d \rangle, \\ &= (AX^h \pmod{g}) + \langle g, Xg, \dots, X^{d-\deg g}g \rangle. \end{aligned}$$

Have a table of integers associated to  $B$ 's, and add  $\deg g$  to all entries meeting  $\mathcal{S}(g)$ . At the end, entries with highest values are likely to yield good pairs.

# The sieving range



Typically, this sieving task is distributed across a huge number of machines, of different computational power. We want to bring more flexibility to this scheme.

# An improvement to the standard sieve: sieve grouping

---

We no longer fix all the bits of  $A$ , but allow some ( $\epsilon$ ) of them to **vary**.

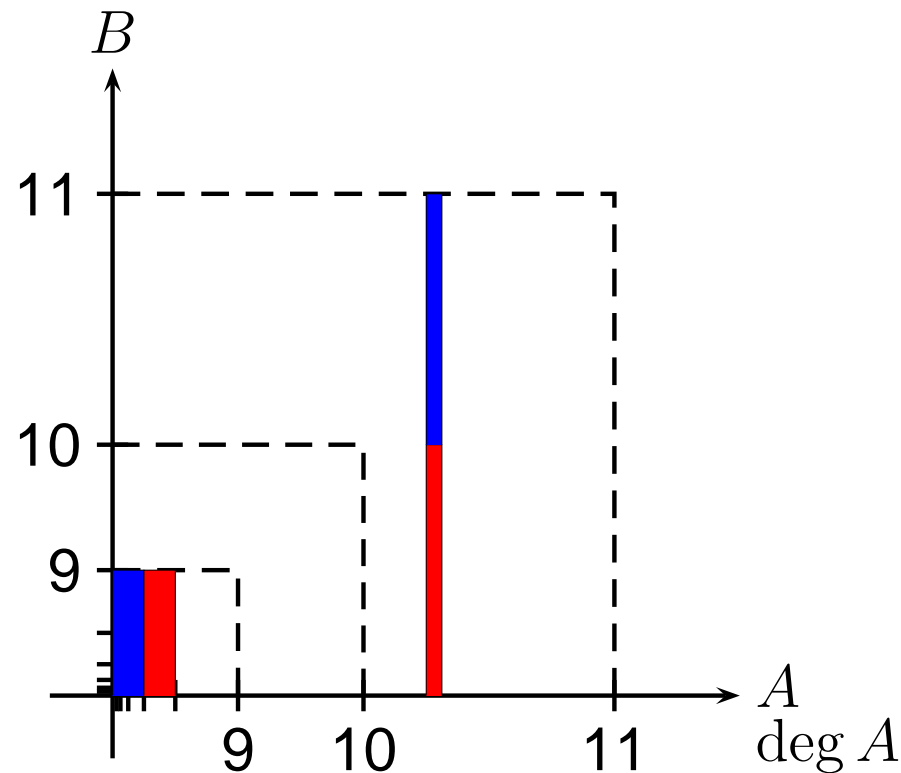
Call  $A_f$  the **fixed part** of  $A$ .

Write  $A = A_f + A_v$ ,  $X^\epsilon | A_f$ ,  $\deg A_v < \epsilon$ . For each  $g$ , we consider:

$$\begin{aligned} \mathcal{S}(g) &= \{(A_v, B) \text{ s.t. } C = (A_f + A_v)X^h + B \equiv 0 \pmod{g}\}, \\ &= \text{affine subspace of } \langle 1, X, \dots, X^d, X^h, \dots, X^{h+\epsilon-1} \rangle, \\ &= (A_f X^h \pmod{g}) + \\ &\quad \langle g, Xg, \dots, X^{d-\deg g}g, X^h + (X^h \pmod{g}), \dots \rangle. \end{aligned}$$

$\mathcal{S}(g)$  is almost as **easy to compute** as previously, but provides initialization data for  $2^\epsilon$  as much sieve space.

# The sieving range with sieve grouping



- Start with **the most interesting part** of the sieve range.  
No loss of time on the less interesting part.  
 $\Rightarrow$  more flexibility, better usage of the sieve range.
- Large RAM**  $\Rightarrow$  reduce the initialization cost to almost nothing.  
It pays off to set up initialization data for  $\epsilon > 0$ , and then do **sub-sieves**.

# Many other sieve improvements are possible

---

- Use of **large primes** (1, 2, more): “Quasi-relations” involving large cofactors can be combined to yield plain smooth relations. We used **double large prime variation**. Complications arise because “large primes” must be considered with their exponent.
- **Avoid sieving** for too small or too large  $g$ 's, to cut down on the sieving and initialization time. We only sieved for  $10 \leq \deg g \leq 22$ .
- Do **special-q** (lattice) sieving: use linear algebra over  $\mathbb{F}_2$  to compute the initialization of the sieve in that case (not extensively used here).
- Sieve over  $D$  if useful (it turned out not to be).  
Initialization harder  $\Rightarrow$  Sieve grouping pays off even more then.

# 4. The resulting linear system

We have a collection of relations of the form:

$$\sum_i (f_i - ke_i) \log \pi_i \equiv 0 \pmod{q = 2^n - 1}.$$

$\Rightarrow$  singular  $N \times N$  matrix  $M$ . Look for  $w \in (\mathbb{Z}/q\mathbb{Z})^N$ ,  $Mw = 0$ .

This matrix is **large** and **sparse**: Few ( $\gamma \ll N$ ) non-zero coefficients per row.

- Gaussian elimination:  $O(N^3)$ .
- Dedicated algorithms:  $O(\gamma N^2) \Rightarrow$  **must** be employed.

Two important stumbling blocks:

- The base ring is very large:  $\mathbb{Z}/(2^n - 1)\mathbb{Z}$
- Not all the coefficients are  $\pm 1$  ( $k$  is 4).

# Dealing with the sparse matrix

---

First pass: **structured gaussian elimination**. This procedure “crunches” the matrix, trying to keep it as sparse as possible.

- An **estimate of the cost** of handling the resulting (still sparse) matrix afterwards is kept, in order to have this estimate minimized.
- While doing this structured gaussian elimination, some rows are removed from the system. **Heuristics** on how to do this were carefully analyzed, and the influence of this on the resulting matrix is **very sensible**.
- Not all coefficients are  $\pm 1$ , so pivoting must avoid excessive coefficient growth.

# 5. Using and improving the block Wiedemann algorithm

---

After the structured gaussian elimination, the matrix is still **sparse**.

Dedicated algorithms available to compute an element of the nullspace:

- (block) Lanczos.
- (block) Wiedemann.

We used the **block Wiedemann** (BW) algorithm.

BW accomplishes a step towards **partial distribution** of the linear algebra. However, there is still a sequential step (**improved** here).

The computation of an element of the nullspace is done modulo each of the large prime factors of  $2^n - 1$ , and the result is recovered by Chinese remaindering.

# Principle of the Wiedemann algorithm

- Given a singular matrix  $M$  of size  $N \times N$  over  $K$ , choose random vectors  $x, y \in K^N$ , and compute:

$$A(X) = \sum_{i=0}^{2N-1} a_i X^i = \sum_{i=0}^{2N-1} (x^T M^{i+1} y) X^i \in K[X].$$

- Try to compute a **rational fraction expansion** of  $A(X)$ , via HGCD or Berlekamp-Massey algorithm.

$$A(X)F(X) = G(X) + O(X^{2N}), \quad \deg F \leq N, \quad \deg G < N.$$

$\hat{F}(X)$  (the reciprocal of  $F$ ) is then a divisor of the minimum polynomial of  $M$ , and  $\hat{F}(M)y$  is in the nullspace of  $M$ .

**Complexity:**  $2N + N = 3N$  matrix-times-vector products,  $+O(N^2)$  or less for the Berlekamp-Massey algorithm.

# Principle of the **block** Wiedemann algorithm

- Replace vectors  $x, y$  by **blocks** of vectors  $x \in K^{N \times m}, y \in K^{N \times n}$ , with  $m$  and  $n$  two chosen integers.
- $A(X)$  becomes a matrix of polynomials:

$$A(X) = \sum_{i=0}^L a_i X^i = \sum_{i=0}^L (x^T M^{i+1} y) X^i \in K^{m \times n}[X], L = \frac{N}{m} + \frac{N}{n} + \epsilon.$$

- We still look for  $F(X) \in K[X]^n$  such that:

$$A(X)F(X) = G(X) + O(X^L), \deg F \leq \frac{N}{n}, \deg G < \frac{N}{n}.$$

- As before, once  $F(X)$  is obtained, an element of the nullspace can be found.

What are the differences with the “plain” Wiedemann algorithm?

# Advantages of BW over plain Wiedemann (1)

---

Computation of  $A(X) = \sum_{i=0}^L (x^T M^{i+1} y) X^i$  is **easier**.

Column  $k$  of  $A(X)$  depends uniquely on column  $k$  of  $y$ .

If computers  $C_0, \dots, C_{n-1}$  each hold a copy of  $M$  and one column  $y_k$  of  $y$ , computer  $C_k$  iterates:

- $y_k \leftarrow M y_k$ .
- Output  $x^T y_k$ .

These computations can take place on different computers, with **zero communication**.

# Advantages of BW over plain Wiedemann (2)

Computation of  $F(X)$  s.t.  $A(X)F(X) = G(X) + O(X^L)$  is **harder**.

• **Matrix** context:

•  $A(X)$ : an  $m \times n$  matrix of polynomials:  $A(X) \in K[X]^{m \times n}$

•  $F(X)$ : an column vector of  $n$  polynomials:  $F(X) \in K[X]^{n \times 1}$

•  $G(X)$ : an column vector of  $m$  polynomials:  $G(X) \in K[X]^{m \times 1}$

⇒ We need a matrix version of the Berlekamp-Massey algorithm.

• **Sequential** nature apparently unavoidable.

• Base field is  $\mathbb{F}_p$ , not  $\mathbb{F}_2$  ⇒ much harder.

Coppersmith suggested one matrix generalization of Berlekamp-Massey, in time  $O(N^2)$ .

We provide an **asymptotically fast, FFT-based extension**, in  $O(N \log^2 N)$ . It turned out to be a big improvement.

# Subquadratic matrix Berlekamp-Massey (1)

We consider “candidate” solutions, for increasing values of  $t$ :

$$A(X)F(X) = G(X) + X^t E(X), \quad \deg F \leq \delta, \quad \deg G < \delta.$$

- Our goal:  $t \geq L$  and  $\delta \leq \frac{N}{n}$ .
- For each  $t$ , a tuple of  $m + n$  “candidates”:  $(f_j, g_j, e_j, \delta_j)_{j=0 \dots m+n-1}$  gathered to form **matrices**:  
 $f(X) \in K[X]^{n \times (m+n)}$   
 $g(X) \in K[X]^{m \times (m+n)}$   
 $e(X) \in K[X]^{m \times (m+n)}$
- Setup: technical but easy,  $t_0 \in O(1)$ , and  $\forall j, \delta_j = t_0$ .

Going from  $t$  to  $t + 1$ :

$$e(0) \text{ known } \rightsquigarrow P^{(t)} \text{ of degree 1 s.t.} \quad \begin{aligned} f^{(t+1)} &= f^{(t)} P^{(t)} \\ g^{(t+1)} &= g^{(t)} P^{(t)} \\ e^{(t+1)} &= e^{(t)} P^{(t)} \frac{1}{X} \end{aligned}$$

We can work with  $e(X)$  only

# Subquadratic matrix Berlekamp-Massey (2)

Fact: Partial knowledge of  $e(X)$  (e.g.  $e(X) \bmod X^b$ ) is enough to:

- compute  $P^{(t)}(X)$ ,
- deduce  $e^{(t+1)}(X) \bmod X^{b-1}$ , since  $e^{(t+1)} = e^{(t)} P^{(t)} \frac{1}{X}$ .

⇒ **Divide-and-conquer** approach. To compute  $f^{(L)}(X)$ :

- Compute  $e^{(0)}(X) \bmod X^L$ , go into recursive procedure:
  - Using  $e^{(0)}(X) \bmod X^{\frac{L}{2}}$ , compute recursively  $P^{(0)} \dots P^{(\frac{L}{2}-1)}$ ,
  - Deduce  $e^{(\frac{L}{2})}(X) \bmod X^{\frac{L}{2}}$ , compute recursively  $P^{(\frac{L}{2})} \dots P^{(L-1)}$ .
  - Compute  $P^{(0)} \dots P^{(L-1)}$
- Then  $f^{(L)}(X) = f^{(0)}(X) P^{(0)} \dots P^{(L-1)}$ .

Multiplication of large polynomial matrices done efficiently with **FFT**.

## 6. The record computation over $\mathbb{F}_{2^{607}}$

---

Factor base size: 766,150 polynomials ( $b = 23$ ).

Sieving code:

- Idle time was used on a large ( $\sim 100$ ) number of linux PCs and alphas (Typical machine: Pentium II 450 MHz).
- Code was run from late-2000 to mid-2001.  
19,000 MIPS years equivalent (?)
- **Double large prime** variation used
- More than 10GB of partial relations gathered.
- At the end, 1,033,593 relations obtained, with more that 80% from large primes.

# The linear algebra step for $\mathbb{F}_{2^{607}}$

---

Linear system obtained: 1,033,593 equations, 766,150 unknowns,  
~ 70,000,000 non-zero coefficients.

After the **structured gaussian elimination** (a few hours CPU):

- Matrix of size  $480,000 \times 480,000$ ,
- 107 coeffs/row on average.
- RAM storage for the matrix: 400MB.

**Block Wiedemann** algorithm done using several SMP machines:

- We used several **unconnected** machines
- We **also** used the **SMP** capabilities of each machine (POSIX threads, etc. . .)

.... / ....

# The linear algebra step for $\mathbb{F}_{2^{607}}$ (continued)

---

- Non-sequential steps: **18 days** on 6 4-cpu 667MHz EV67's
- Sequential step: **4 days** on one 667MHz EV67 cpu.
  - Nearly 2 months of computation time saved here with our new recursive approach.
  - Enormous polynomial matrix multiplications with FFT required very large swap memory (20GB).
- Interest of having used the **block** algorithm: total time of 3 weeks instead of 4 months !

The computation of an **individual log** takes 1 to 2 hours CPU on a 700MHz Pentium III.

## 7. Conclusion

---

- A **battery of improvements** have been used to improve the sieve part: sieve grouping, incomplete sieves (skipping low and high degree  $g$ 's), (double) large primes, special-q sieving.
- Some aspects have not been used, or not extensively: multiple ( $> 2$ ) large primes schemes, special-q sieving
- The linear algebra benefited from heuristics for the structured gaussian elimination, and an improvement of the block Wiedemann algorithm.
- This is a very long computation, involving a fair amount of both **human time** and **computational power**.
- Access to a private, closed network of computers is bound to reduce drastically the computation time.