

Algorithms for discrete logarithms in finite fields and elliptic curves

ECC "Summer" school 2015

E. Thomé



```
/* CARMEL */
/* C.A.
   *M.
   *L.
   */
d[S],d[0]
(i,j);a=mod(x-
++i+d; ++q[ i+1; A],B=
M[i]; for(i= for(i
--i; ++i; }get
+q[i-; M[i] L[A]
E=1;L=M,a=;D= s+E*B*M=L;L=(M+E
IA,E=CA+a --[a]);print
/* cc carmel.c; echo 23 22 21 20 p | ./a.out */
C.A.
M.
L.
for
++d;
L[0]?
+M
D[A]
for
++L;
/*
+eB
M/2
+M);
```

Sep. 23rd-25th, 2015

Part 1

Context and old algorithms

Context, motivations

Exponential algorithms

$L(1/2)$ algorithms

Plan

Context, motivations

Exponential algorithms

$L(1/2)$ algorithms

Plan

Context, motivations

Definition

What is hardness?

Good and bad families – should we care only about EC?

Cost per logarithm

The discrete logarithm problem

In a cyclic group, written multiplicatively

- $(g, x) \rightarrow g^x$ is easy: polynomial complexity;
- $(g, g^x) \rightarrow x$ is (often) hard: **discrete logarithm problem**.

For an elliptic curve E , written additively:

- $(P, k) \rightarrow [k]P$ is easy;
- $(P, Q = [k]P) \rightarrow x$ is hard.

Cryptographic applications rely on the **hardness** of the discrete logarithm problem (DLP).

Another view on the DLP

In case the group we are working on is not itself cyclic, DLP is defined in a cyclic sub-group (say of order n).

Thus we know that we have an **isomorphism**.

$$\begin{aligned}\mathbb{Z}/n\mathbb{Z} &\rightarrow G, \\ k &\mapsto g^k\end{aligned}$$

DL is the direction \leftarrow . This map is **not computationally explicit**.

DH versus DL

There are several ancillary problems to the DLP, the closest ones being the (computational) Diffie-Hellman problem:

$$\text{CDH: } (g, g^x, g^y) \rightarrow g^{xy}$$

and the decisional Diffie-Hellman problem:

$$\text{DDH: } (g, g^x, g^y, h) \rightarrow h \stackrel{?}{=} g^{xy}$$

Crypto protocols rely on the hardness of some specific problems.

- DH key exchange relies on CDH;
- El Gamal encryption relies on CDH;
- For some primitives, semantic security may rely on DDH;
- ...

Caveat: many newly invented cryptographic protocols come with their new purportedly hard problem.

DH versus DL

- **Trivial:** can solve DLP \Rightarrow can solve CDH \Rightarrow can solve DDH.
- The converse is “often” true [Mau94, MW96, MSV04].

All NAME-YOUR-DL-RELATED-HARDNESS-ASSUMPTION-HERE instances can at worst be broken by solving DL.

- In many cases, this is the best solution known... not always.

What we won't do

We do **not** address here the DH-related problems, but really the computation of DL.

Plan

Context, motivations

Definition

What is hardness?

Good and bad families – should we care only about EC?

Cost per logarithm

Hardness

Key notion

What does it mean when we say:

- DLP is hard on elliptic curves.
- DLP is moderately hard on finite fields.
- DLP is easy on small characteristic finite fields.

Hardness

A family of groups \mathcal{G}

Size parameter $\lambda \rightarrow \mathcal{G}(\lambda) =$ a set of groups of size λ .

Here, size $\lambda = \log_2 \#G$ (roughly).

Some basic requirements:

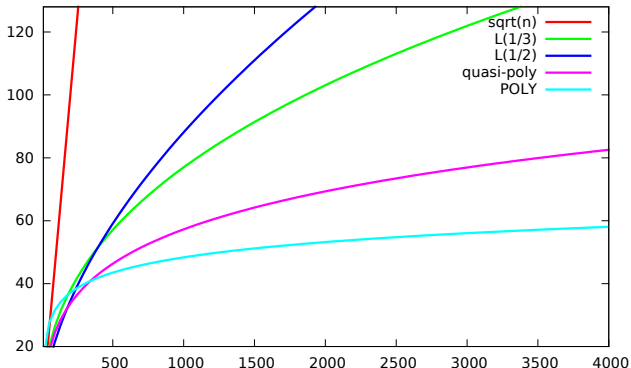
- Counting $n = \#G$ for $G \in \mathcal{G}(\lambda)$ must be feasible;
- We want $O(\lambda)$ bits for storing elements;
- Arithmetic in G must be reasonably efficient, $\text{POLY}(\lambda)$.

Examples: finite fields; binary finite fields; prime fields; elliptic curves over binary fields; . . .

Hardness: we want the cost of computing DL by most efficient means known to **grow fast**.

Hardness w.r.t input size

Plotting $\log(\text{computational difficulty})$ as a function of $\log(\#G)$.

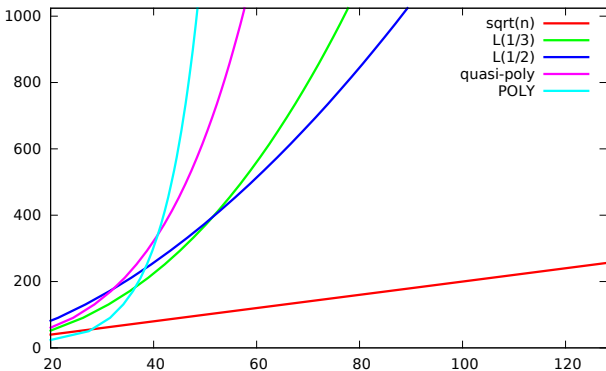


Slow algorithms get expensive pretty soon.

Key size w.r.t desired security / time

Feasibility limit = computational resources; grows with **time** (Moore's law).

- **security parameter** = $\log(\text{attack cost})$ is a measure of **time**.
- key size to be chosen appropriately.



Plan

Context, motivations

Definition

What is hardness?

Good and bad families – should we care only about EC?

Cost per logarithm

Which groups shall we avoid ?

Here are some group families and rough DL hardness:

- Easy DL: **truly bad for crypto**: $(\mathbb{Z}/N\mathbb{Z}, +)$;
- Not-so-hard DL: **finite fields**, curves of very large genus, class groups of number fields;
- Hard DL: **elliptic curves**, **curves of genus 2**.

And beyond that, groups whose order is a product of small primes are a waste, because of **Pohlig-Hellman reduction**.

Why care about FF ?

Topic here

DLP in [finite fields](#) and [elliptic curves](#).

Elliptic curves enjoy [harder DLP](#) than finite fields.

Why don't we ditch finite fields altogether ?

Reach of studying FF DLP

Finite fields are not ideal for crypto. Why is it interesting at all ?

- It's a basic object. Is worth looking at mathematically speaking.
- It's a basic object. It's been there from the start.
- It's a basic object. Some crypto protocols prefer these.
- Pairings.
- Tricks with subgroups of the multiplicative group.

Pairing-based crypto

An indirect reason to look at FF-DLP

Some (\pm exotic) cryptographic protocols use **pairings**:

$$e : \mathbb{G} \times \mathbb{G} \rightarrow K^*, \quad (K \text{ finite field})$$

- Identity-based encryption [BF01],
- Tripartite DH [Jou00],
- Short signatures [BLS04],
- ...

The finite field is intrinsically there; cannot be replaced.

Known practical situations:

- \mathbb{G} = an elliptic curve over \mathbb{F}_q ,
- K a **small extension** of \mathbb{F}_q .

Why do subgroup DLP ?

There's a small catch with the Pohlig-Hellman reduction.

- Computing the log in G (thus modulo n) might use some special structural properties of G (e.g. stability under some extra operations).
- Computing the log in a subgroup might fail to exploit these properties any better.

Subgroups of finite fields

Consider $G = \mathbb{F}_p^\times$, and H a subgroup, $\#H = q \mid (p - 1)$.

- DLP algorithms for \mathbb{F}_p^\times lead to choose $p > 2^{2048}$ (so to say).
- Practical attacks for DLP in H , beyond DLP in \mathbb{F}_p^\times , are generic ones. The safety condition on $q = \#H$ is less stringent (say $q > 2^{256}$).

Subgroups of finite fields

Some protocol designs exploit this subgroup trick.

- DSA takes a $q \approx 160$ -bit subgroup of \mathbb{F}_p^\times , with 1024-bit p .
Signature data is mod q , not mod p .
- The XTR cryptosystem [LV00] uses a reduced representation of elements within a subgroup of \mathbb{F}_{p^6} .
- Torus-based cryptography [RS08]: generalization of similar principle.

What this implies for DLP challenges:

- Imagine $\#\mathbb{F}_{p^{2048}}^\times = 2 \times p_{256} \times p_{1792}$; DLP in the 1792-bit subgroup is hardly relevant for crypto.
- Pairing-based crypto example $E \times E \rightarrow \mathbb{F}_{q^k}$.
Only DLP in subgroup of order $\#E$ is interesting.

Subgroups of finite fields (summary)

Bear in mind

- we are not necessarily interested in the DLP in the whole multiplicative group *per se*;
- we shall not be annoyed by some uninteresting prime factors behaving oddly.
- DL modulo small primes can be handled differently anyway.

Plan

Context, motivations

Definition

What is hardness?

Good and bad families – should we care only about EC?

Cost per logarithm

The per-log cost issue

Context for finite field DLP

$K = \mathbb{F}_{p^n}$ a finite field; $G = \langle g \rangle$ a subgroup of K^\times .

We'll see:

- Basic algorithms which apply to general groups (incl EC);
- Algorithms specialized for the finite field context.

We might regard the DLP in two possible ways.

- Given $a = g^x \in G$, compute x ; just once.
- Do **some precomputation**, producing data so as to be able to compute many log at smaller cost.

There's a preference for the latter (look at LOGJAM, for instance).

The art of FF-DLP

Algorithms specialized for finite field DLP share background with:

- Algorithms for factoring integers.
Much of the work which led to DLP algorithms initially came from the factoring business.
- DLP computation of high genus curves [EGT11].

Plan

Context, motivations

Exponential algorithms

$L(1/2)$ algorithms

Plan

Exponential algorithms

Baby-step–Giant-step and ρ

Parallel collision search

Some ECDLP records

Generic algorithms (work in all cases)

First focus on DLP algorithms for generic groups. This means we cannot use specific properties of G , just group operations. The group acts as a **black box**.

Known generic solutions to DLP (here $n = \#G$):

- Enumeration: $O(n)$;
- Baby-step–Giant-step: deterministic time and space $O(\sqrt{n})$;
- Pollard ρ : probabilistic time $O(\sqrt{n})$, space $O(1)$.
- Parallel collision search (λ): same context, in parallel.

These are **exponential** algorithms in the **bit-size of G** .

Shanks: Baby-step–Giant-step

One writes $\log_g a = uC + v$, $0 \leq v < C$, $0 \leq u < n/C$

$$\text{Then: } g^{uC+v} = a \Leftrightarrow a(g^{-C})^u = g^v.$$

Baby steps: compute $\mathcal{B} = \{g^v, 0 \leq v < C\}$;

Giant steps: ● compute $h = g^{-C}$;

● for $u = 0..n/C$, if $ah^u \in \mathcal{B}$, then stop.

End: $ah^u = g^v$ for some u, v , hence $\log_g a = uC + v$.

Analysis: $C + n/C$ group operations and n/C membership tests.

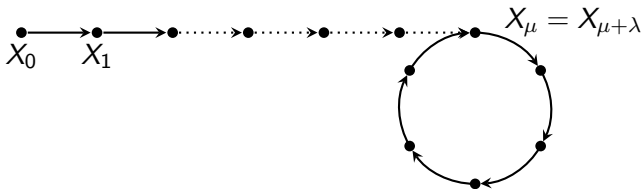
Deterministic time and space $\tilde{O}(\sqrt{n})$ for $C = \sqrt{n}$.

Implementation: ● membership testing with hashing;

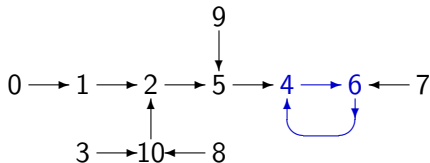
● all kinds of trade-offs possible if low memory.

Pollard's ρ

Prop. Let $f : E \rightarrow E$, $\#E = m$; $X_{n+1} = f(X_n)$ with $X_0 \in E$. The functional graph of X is:



Example. Let $E_m = \mathbb{Z}/11\mathbb{Z}$, $f : x \mapsto x^2 + 1 \pmod{11}$:



Application to discrete logarithms

Say we look for k such that $Q = [k]P$, with $P, Q \in E(\mathbb{F}_q)$.

A way to create “random-looking” functions

- For $1 \leq k \leq r$, pick random integers γ_k and δ_k , and precompute $M_k = [\gamma_k]P + [\delta_k]Q$;
- define an arbitrary function $\mathcal{H} : E \rightarrow \{1, \dots, r\}$;
- define $f(Y) = Y + M_{\mathcal{H}(Y)}$.

Experimentally, for $r \geq 20$, we get f “sufficiently random”.

By iterating $X_{k+1} \leftarrow f(X_k)$, one keeps track of:

- $X_k \in G$;
- Integers u_k and v_k in $\mathbb{Z}/n\mathbb{Z}$ such that $X_k = [u_k]P + [v_k]Q$.

Whenever $X_m = X_n$, we unveil k (with good probability).

Analyzing Pollard's ρ

How long before $X_m = X_n$?

Standard analysis tools (Flajolet-Odlyzko) allow to give expected values for m and n .

- $E[m] = E[n] = \sqrt{\pi/8 \cdot \#E(\mathbb{F}_q)}$;
- Variance and more also known.

In practice, look for $X_{2t} = X_t$. $E[t] = \sqrt{\pi^5/288 \#E(\mathbb{F}_q)}$;

Floyd's algorithm:

```
X ← X0; Y ← X0; e ← 0;
repeat
X ← f(X); Y ← f(f(Y)); e ← e+1;
until X = Y;
```

Plan

Exponential algorithms

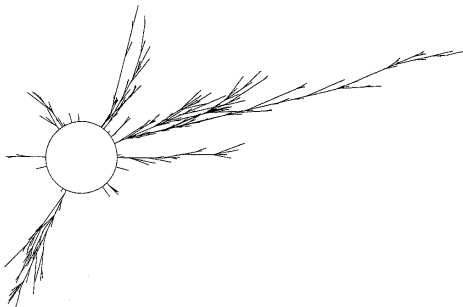
Baby-step–Giant-step and ρ

Parallel collision search

Some ECDLP records

Searching for collisions in parallel

The “big picture” of the functional graph of f looks like this:



- Principle: have several workers start at different random points in the graph.
- Challenge: how to detect collisions efficiently ?

The parallel collision search method

Foklore idea, attributed to many ([QD90, vOW99]).

- Node K iterates $x \leftarrow f(x)$.
- Too expensive to check **every** x will all points computed elsewhere.
- Only rare **distinguished points** are considered for checking: e.g. if the 20 lowest bits of their hash are zero.
- DPs are stored on a server.
- Caveat: avoid short cycles. Node restart from a new fresh random point quite often.

The complexity in total is still $O(\sqrt{n})$, and this scales up very well.

Is that all ?

We've seen two algorithms which work on generic groups, and solve DL in $O(\sqrt{n})$.

Can we do better ?

- Not for **generic** groups.

Nechaev-Shoup [Nec94, Sho97]

For a **generic** (black-box) group G with $n = \#G$, no discrete logarithm algorithm succeeds in time less than $\Omega(\sqrt{n})$.

- Caveat: generic groups do not exist.

Plan

Exponential algorithms

Baby-step–Giant-step and ρ

Parallel collision search

Some ECDLP records

Computational feats

Generic algorithms are the best ones known for DLP on elliptic curves. Thus the most intensive calculations are in this context.

All used some form of parallel collision search.

- ~ 2000: several certicom ECC challenge “exercises” done by R. Harley. Software still online.
http://crystal.inria.fr/~harley/ecdl_top
- 2002, 2004: distributed effort, 109-bit both prime and binary.
- 2009: 112-bit prime ECDLP using Playstations (EPFL).
http://laca1.epfl.ch/112bit_prime
- 2014, 2015: FPGA records for ECDLP over 113-bit binary fields (one Koblitz one non-Koblitz curves).
- 2009- effort to attack a 131-bit Koblitz curve (not completed, probably stopped). <http://ecc-challenge.info/>

Plan

Context, motivations

Exponential algorithms

$L(1/2)$ algorithms

Plan

$L(1/2)$ algorithms

Smoothness, and combination of congruences

Adleman's algorithm

Analysis of Adleman's algorithm

Adaptation – Other $L(1/2)$ algorithms

Combination of congruences

As it turns out, **finite fields** have faster DLP algorithms than generic groups.

- First algorithm of this kind: Adleman [Adl79], based on the **combination of congruences** idea (cf factoring).
- Analysis requires some standard analytic number theory tools.
- More advanced algorithms followed.

Smoothness

We often talk about **smoothness** of polynomials or integers.

Definition: smoothness

- A polynomial of degree at most n is k -smooth if all its irreducible factors have degree at most k .
- An integer $\leq X$ is B -smooth if all its prime factors are $\leq B$.

Probability of smoothness is controlled essentially by the **size ratio**:

$$u = \frac{n}{k}, \text{ or } u = \frac{\log X}{\log B}.$$

Simple-to-remember: the probability is roughly $u^{-u(1+o(1))}$.

- Canfield-Erdős-Pomerance + many variants.
- Some caveats with limit cases.
- Analytic number theory can give much stronger results.

About smoothness

The cost of **smoothness testing** depends on which kind of object we're talking about.

- Easy for **polynomials**, just like factoring polynomials, has polynomial complexity.
- For **integers**, it is trickier, as factoring integers is hard. It's possible to work around this difficulty both theoretically and in practice.

Not always a proper notion of smoothness

Smoothness = **construction kit** situation.

- Big pieces made of small pieces.
- Works fine for \mathbb{Z} or $\mathbb{F}_p[x]$, much less for EC.

Plan

$L(1/2)$ algorithms

Smoothness, and combination of congruences

Adleman's algorithm

Analysis of Adleman's algorithm

Adaptation – Other $L(1/2)$ algorithms

Adleman's algorithm

Aim at $\log_g h$ in \mathbb{F}_p^\times . Let $B \approx 2^\beta$ be a bound.

Key choice: a **factor base** = set of privileged elements

$$\mathcal{F} = \{\pi \bmod p, \pi \text{ prime} \leq B\}.$$

- Whenever $g^{\text{random}} \bmod p$ is **smooth**, we have

$$\begin{aligned}g^r &\equiv p_1^{e_1} \cdots p_K^{e_K} \pmod{p}, \\r &\equiv e_1(\log p_1) + \cdots + e_K(\log p_K) \pmod{p-1}\end{aligned}$$

with $\{p_i\} \subset \mathcal{F}$.

- By **linear algebra** we solve for the unknowns $(\log p_i)$.

Next, given the challenge h :

- Compute random elements $b = hg^r$ until b is B -smooth.
- Use the precomputed $(\log p_i)$'s to express $\log h$.

Adleman's algorithm

- Reduce g^{random} mod p ;
- Hope for B -smoothness.
- Yields relation between FB elements.
(additive relation with their log's)
- Individual log: compute $g^{\text{random}} h$.

\mathbb{Z}
↓
 \mathbb{F}_p

smoothness

relation

The linear algebra system showing up

Characteristics of the system

- Equations are $r \equiv e_1(\log p_1) + \cdots + e_K(\log p_K) \pmod{(p-1)}$.
- Unknowns are $(\log p_i)$.
- Matrix coefficients are e_j .
- Each matrix row has **few coefficients**, which are **small integers**.
- The system is defined over $\mathbb{Z}/(p-1)\mathbb{Z}$.

The algorithmic tools to solve such systems are **sparse linear algebra algorithms** (which are much more developed now than they were in 1970's).

- Will tell more about these later;
- Good to know: nowadays we do have the tools to do this in **quadratic time**.

Plan

$L(1/2)$ algorithms

Smoothness, and combination of congruences

Adleman's algorithm

Analysis of Adleman's algorithm

Adaptation – Other $L(1/2)$ algorithms

Size matters

Recall $p \approx 2^\lambda$ and $B \approx 2^\beta$.

The set of primes below B is called **the factor base**.

Which size for the factor base elements ? (= which B ?)

- too small (= too few)?
 - Will have a hard time finding relations.
 - Cheap linear algebra.
- too large (= too many)?
 - Relations become cheaper.
 - Linear algebra becomes an obstacle.

Analysis

Recall $p \approx 2^\lambda$ and $B \approx 2^\beta$.

- $\#\{\text{primes below } B\} = B / \log B$, not that far from $B = 2^\beta$.
- $a = g^r$ is an λ -bit integer.
Smoothness probability: $(\lambda/\beta)^{-\lambda/\beta(1+o(1))}$.
We need 2^β relations.
- Linear algebra cost is polynomial in B .

Optimal choice: $\beta \approx \sqrt{\lambda}$.

The overall complexity is $\exp(O(\sqrt{\lambda \log \lambda}))$.

- This is called a **sub-exponential complexity**.
- MUCH better than $\sqrt{p} = \exp(\lambda/2)$

Consequence: key size \approx square of time (see slide 10).

Precomputation vs individual logarithms

Note: as presented, Adleman's algorithm has:

- a **precomputation** stage: logs of FB elements.
- an **individual log** stage: given a , find $\log_g a$.

One can show that the latter is cheaper.

Nice DL algorithms nowadays keep this small per-logarithm cost.

The L function

Handy function (R. Schroepel and/or C. Pomerance):

$$L_x[\alpha, c] = \exp\left(c(\log x)^\alpha (\log \log x)^{1-\alpha}\right).$$

- $L_x[0, c] =$ polynomial in $\log x$.
- $L_x[1, c] =$ exponential in $\log x$.
- L is often called the **sub-exponential function**.

Usual shorthand in these slides: $L_x[\alpha]$ denotes $L_x[\alpha, c]$ for some **explicitly computable constant c** .

L function arithmetic

Computation rules with $L_x(\alpha, c)$

$$L_x[a, u] \times L_x[b, v] = \begin{cases} L_x[a, u + o(1)] & \text{if } a > b, \\ L_x[b, v + o(1)] & \text{if } b > a, \\ L_x[a, u + v] & \text{if } a = b. \end{cases}$$

$$L_x[a, u > 0] + L_x[b, v > 0] = \begin{cases} L_x[a, u + o(1)] & \text{if } a > b, \\ L_x[b, v + o(1)] & \text{if } b > a, \\ O(L_x[a, \max(u, v)]) & \text{if } a = b. \end{cases}$$

$$L_{L_x[b, v]}[a, u] = L_x[ab, uv^a b^{1-a} + o(1)].$$

$$L_x[b, v]^{\log_{\log x} L_x[a, u]} = L_x[a + b, uv].$$

Restating CEP

Reformulation of Canfield-Erdős-Pomerance

An integer $\leq L_x(\alpha, u)$ is $L_x(\beta, v)$ -smooth with probability:

$$L_x(\alpha - \beta, -\frac{u}{v}(\alpha - \beta))^{1+o(1)}.$$

For example, a **random** integer modulo N has a probability $L_N(1/2, \cdot)$ of being $L_N(1/2, \cdot)$ -smooth.

Smoothness for integers

Interlude: the **ECM factoring method** unveils a factor p of N in time $L_p[1/2]$. (Idea: set $B_1 = L_p[1/2, c]$. Hope for $\#(E \bmod p)$, which is $L_p[1, 1]$, to be B_1 -smooth.)

Consequence: testing a number $x \approx N$ for smoothness with respect to a bound $B = L_N[\gamma, c]$ costs:

$$L_{L_N[\gamma, c]}[1/2] = L_N[\gamma/2].$$

This may be seen as mostly of theoretical interest, but does save the day when analyzing difficult steps of advanced algorithms.

Analyzing Adleman again

Integers mod p have magnitude $L_p(1, 1)$.

Write the smoothness bound B as $L_p[\gamma, c]$ for some γ, c .

- Linear system cost: $L_p[\gamma, 2c]$;
- Smoothness probability: $L_p[1 - \gamma, 1/c(1 - \gamma)]$.
- Relations needed: $L_p[\gamma, c]$.
- Overall cost: $L_p[\max(\gamma, 1 - \gamma)]$.

This readily gives the optimal $\gamma = 1/2$ (and the lower order term comes, too).

All the subexponential algorithms nowadays are analyzed with this machinery.

Plan

$L(1/2)$ algorithms

Smoothness, and combination of congruences

Adleman's algorithm

Analysis of Adleman's algorithm

Adaptation – Other $L(1/2)$ algorithms

Adaptation to binary fields

Obvious fact: what we have just seen for \mathbb{F}_p is also valid for

$$\mathbb{F}_{2^n} = \mathbb{F}_2[x]/P(x)$$

where smoothness of integers is replaced by smoothness of polynomials, and bit size of integers by degree.

DL in small characteristic finite fields:

- 1982-1983, practical improvements to Adleman's algorithm.
DL in $\mathbb{F}_{2^{127}}$ within reach.
- 1983, 1984: Coppersmith's algorithm.
- 1994, 2000: Function Field Sieve.
- 2013: QPA.

DLP on large genus curves

On curves, it is difficult to come up with a notion of **smoothness** which means something.

Slight exception: **curves of large genus** [ADH94]

Archetypal situation: p fixed, C_g hyperelliptic curve of genus g , study DLP in $J_g = \text{Jac}_{C_g}(\mathbb{F}_p)$ (which has $\#J_g \approx p^g$).

- elements of J_g are represented by **divisors** of weight $r \leq g$;

$$D = (P_1) + \cdots + (P_r) - r(\infty).$$

- divisors of **effective weight** $r \leq \sqrt{g}$ taken as factor base;
- smoothness test = factor $u(t)$ in Mumford representation.
- Probability in $L(1/2)$. **Complete DLP algorithm in $L(1/2)$.**

Other index calculus attempts (NOT $L(1/2)$)

When **some** algebraic structure exists, can we use it to define a factor base ?

- Example: $E(\mathbb{F}_{2^n})$, could imagine $\mathcal{F} = \{P \text{ with "small" } x, \text{ e.g. only lower } \sqrt{n} \text{ coord. non-zero}\}$.
- Use **Semaev summation polynomial** to express the equation:

$$Q = P_1 + P_2 + \cdots + P_n$$

for arbitrary Q , and $P_i \in \mathcal{F}$.

- Finding relations becomes **polynomial system solving**.
- As before, linear algebra to get logarithms of \mathcal{F} .
- Some controversy regarding complexity (see Kesters talk).

For some well-chosen cases of $E(\mathbb{F}_{p^n})$, can reach $L(2/3)$ or $L(3/4)$ complexity (Diem).

Part 2

$L(1/3)$ algorithms

History and setting

General setup – easy example with FFS

NFS-DL

DLP in \mathbb{F}_{p^n}

$L(1/3)$ for large genus, small degree curves

Plan

History and setting

General setup – easy example with FFS

NFS-DL

DLP in \mathbb{F}_{p^n}

$L(1/3)$ for large genus, small degree curves

A bit of history

Enter the realm of [NFS-like algorithms](#).

- 1970's: not many of the advanced factoring methods actually had a complexity analysis.
The first $L(1/2)$ formula appeared late.
- 1980: Adleman's method for computing discrete logarithms was only moderately efficient, and did not seem so much of a threat.
Adapting DLP-based cryptosystems to using \mathbb{F}_{2^n} seemed a reasonable thing to do because of [implementation ease](#).

A bit of history

- 1984: Coppersmith algorithm [Cop84]:
The point of this paper is that this adaptation was ill-advised.
Special algorithm for DLP in $\mathbb{F}_{2^n}^\times$. Complexity $L_{2^n}[1/3, c]$.
- 1989: first works towards the Number Field Sieve for factoring integers. 1993: Lenstra-Lenstra book [LL93].
- 1993: Gordon's algorithm for NFS-DL. *Very awkward* for individual logarithms. Several further works.

A bit of history

- 1984: Coppersmith algorithm [Cop84]:
The point of this paper is that this adaptation was ill-advised.
Special algorithm for DLP in $\mathbb{F}_{2^n}^\times$. Complexity $L_2^n[1/3, c]$.
- 1989: first works towards the Number Field Sieve for factoring integers. 1993: Lenstra-Lenstra book [LL93].
- 1993: Gordon's algorithm for NFS-DL. *Very awkward* for individual logarithms. Several further works.

Coppersmith's algorithm really is a historical mark

- It didn't seem at all clear at the time that Coppersmith's method was going to generalize to broader contexts.
- While Coppersmith's method came with an inventive descent algorithm, crafting this step for cousin DL algorithms took years.

What hinders Adleman's algorithm

The following misfeatures have been improved in further works:

- Elements created as $g^r \bmod p$ are **as large as p** .
All we can say is $a \in L_p[1]$.
- For computing individual logarithms (e.g. when we consider the target h), we have no way to take advantage of the fact that **h might perhaps have not-so-large bit size**.

If we could do the latter, we might imagine a **recursive procedure**:

Large $h \rightarrow$ product of smaller elements \rightarrow yet smaller $\rightarrow \{p_i\}$.

Such a recursive procedure exists in the modern context and is called a **descent**.

Plan

History and setting

General setup – easy example with FFS

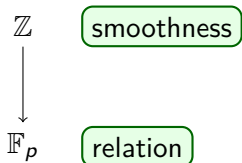
NFS-DL

DLP in \mathbb{F}_{p^n}

$L(1/3)$ for large genus, small degree curves

Adleman's algorithm = starting point

- Reduce g^{random} mod p ;
- Hope for β -smoothness.
- Yields relation between FB elements.
(additive relation with their log's)
- Individual log: compute $g^{\text{random}} h$.

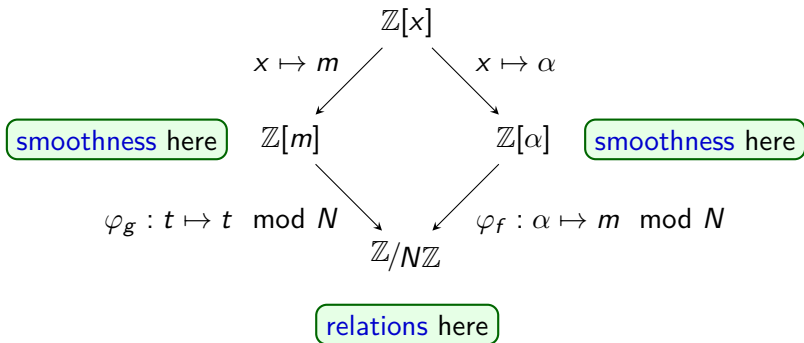


Common subexponential framework

- Collect **relations** between “objects”;
- Define meaningful “logarithms” for these objects;
- Solve a large **linear system**;
- Express **individual logarithms** as combinations of known ones.

Diagram from NFS factoring

This commutative diagram is from the [Number Field Sieve](#) (1993) as a factoring algorithm.



This is still relevant, but before going into detail, we'll look at a specialized example.

The Function Field Sieve

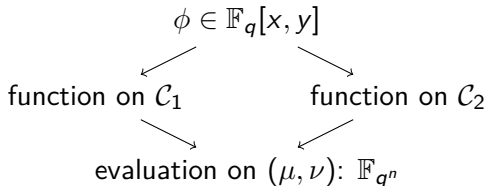
FFS (1994) inherits from the [Number Field Sieve](#) algorithm.

FFS Setting (for \mathbb{F}_{q^n} , q small)

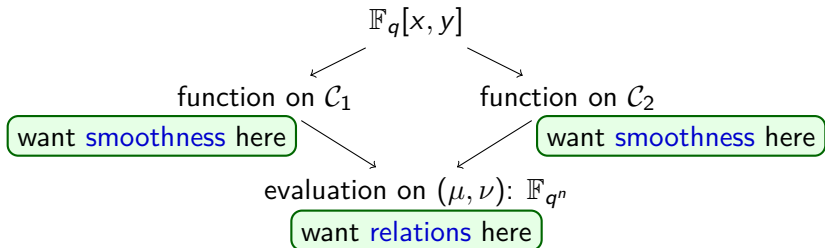
Consider two plane curves defined over \mathbb{F}_q :

$$\mathcal{C}_1 : C_1(x, y) = 0; \quad \mathcal{C}_2 : C_2(x, y) = 0$$

- such that $\mathcal{C}_1 \cap \mathcal{C}_2$ has a point (μ, ν) defining \mathbb{F}_{q^n} ;
- i.e. $\text{Res}(C_1, C_2)$ has an irreducible factor of degree n .



FFS Setting for \mathbb{F}_{q^n} , q small



- We want smoothness on both sides;
 - This occurs in the degree 0 divisor class group of both curves;
 - Down to earth: boils down to some polynomial factoring nicely.
- We get relations from the fact that the diagram commutes.

How much complication this means depends on the curves chosen.

Example: the easy FFS

The FFS scenario is quite flexible, so illustrate with an easy case.

- Our target is \mathbb{F}_{q^n} (bottom of the diagram);
- Any degree n irreducible dividing $\text{Res}(C_1, C_2)$ will do !

Easy choice of curves

$$C_1(x, y) = y - f(x); \quad C_2(x, y) = x - g(y)$$

with f, g such that $g(f(x)) - x$ has a degree n irreducible factor.

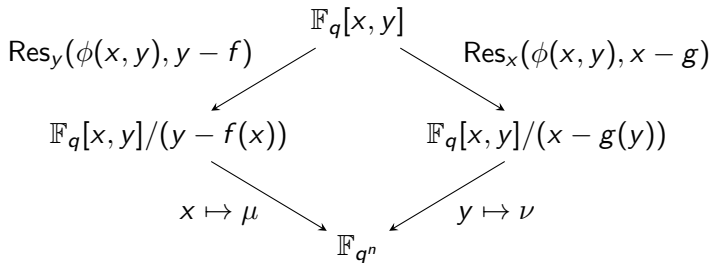
- Let $D(x)$ be this degree n irreducible factor.
- Let μ be a root of D in \mathbb{F}_{q^n} , and $\nu = f(\mu)$:
 - (μ, ν) is clearly on C_1 since $\nu = f(\mu)$.
 - it is also on C_2 since $g(\nu) = \mu$.

Good news ● Curves C_1 and C_2 have genus 0.

- The function field is $\mathbb{F}_q(t)$ in both cases.

The “simple FFS” diagram

We may rewrite the diagram:



Two sides

We look for favorable $\phi(x, y)$ such that:

- On the x (left) side, $\phi(x, f(x))$ is smooth;
- On the y (right) side, $\phi(g(y), y)$ is smooth.

We want **BOTH** to happen at the same time. Few a, b will work.

Factor bases

Implicitly, we are considering two distinct factor bases:

- Irreducible polynomials in x of small degree ($\leq \beta$);
- Irreducible polynomials in y of small degree ($\leq \beta$).

These are distinct because the maps $x \mapsto \mu$ and $y \mapsto \nu$ have distinct images.

Example: the easy FFS

Pick a function $\phi \in \mathbb{F}_q[x, y]$.

- $\phi(x, f(x))$ is a function on \mathcal{C}_1 .

Could be **smooth**: $\phi(x, f(x)) = p_1(x) \cdots p_r(x)$.

- $\phi(g(y), y)$ is a function on \mathcal{C}_2 .

Could be **smooth**: $\phi(g(y), y) = q_1(y) \cdots q_s(y)$.

Doubly smooth ϕ yield **multiplicative relations** in $\mathbb{F}_{q^n}^\times$:

$$p_1(\mu) \cdots p_r(\mu) = q_1(\nu) \cdots q_s(\nu).$$

Work plan:

- Collect many such relations.
- Build linear system with unknowns $\{\log_g p(\mu)\} \cup \{\log_g q(\nu)\}$, with p and q have degree \leq some β .

A quick analysis of FFS

A balanced case for the analysis of FFS:

- Define \mathcal{C}_1 and \mathcal{C}_2 with $\deg f = \deg g = \sqrt{n}$;
- Pick functions ϕ with $\deg_x \phi = \deg_y \phi = \sqrt[6]{n}$
 $\Rightarrow \deg_x \phi(x, f(x)) = \deg_y \phi(g(y), y) \approx n^{2/3}$.
- Define a smoothness degree $\beta = \sqrt[3]{n}$.
- Probability of smoothness in $L_{q^n}(1/3)$.
- Complexity of the whole algorithm is in $L(1/3)$.

Many variants exist. Computation of individual algorithms with a [descent](#) procedure.

Analysis

Collecting all the degree info we have suggested:

- Factor bases = irreducibles of degree $\leq n^{1/3}$; $L[1/3]$
- We have chosen $\deg_{x,y} \phi = n^{1/6}$; $L[1/3]$
- Resultants on both sides have degree $n^{2/3}$; $L[2/3]$
- Smoothness probability is thus $L_{q^n}[1/3]$. $L[1/3]$
- Collecting $L_{q^n}[1/3]$ relations takes $L_{q^n}[1/3]$ (just exhausting the search space). $L[1/3]$
- Linear algebra takes $L_{q^n}[1/3]$. $L[1/3]$

The magic here comes from the way we create relations.
This yields a drop from $L(1/2)$ to $L(1/3)$.

Plan

History and setting

General setup – easy example with FFS

NFS-DL

DLP in \mathbb{F}_{p^n}

$L(1/3)$ for large genus, small degree curves

Plan

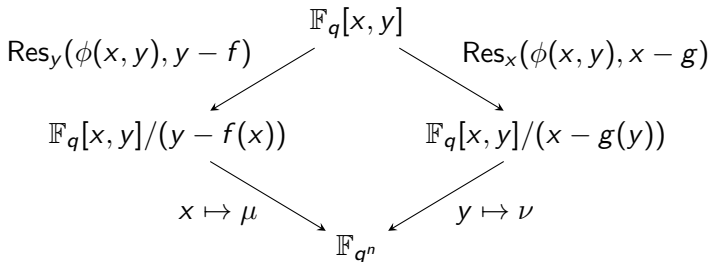
NFS-DL

NFS-DL setting and headaches

Rough analysis

The “simple FFS” diagram

Recall the diagram we had.



What to reuse from our simple FFS example

Our toy FFS example has some peculiarities:

- These **two roles** of x and y are really situation specific;
- The existence of **two sides**, however, is common with other cases;
- The **diagram** will remain;
- The **general algorithm framework** will remain;
- Unfortunately objects become **more complicated** than mere polynomials.

Of course, for attacking DL in \mathbb{F}_p , we will talk more about **integers** than **polynomials**.

Strategy

Goal: create relations mod p as **images** by **ring morphisms** from two different structures, linked by a commutative diagram.

NFS-DL: ● these ring morphisms come from **number fields**
● usually, we take one of these number fields to be \mathbb{Q} .

This is exactly the same framework as for NFS (for factoring).

The NFS-DL setup

We have:

- a number field $K = \mathbb{Q}(\alpha)$ defined by $f(\alpha) = 0$, for f irreducible over \mathbb{Q} and $\deg f = d$;
- another irreducible polynomial g such that f and g have a common root $m \bmod p$ (example: $g = x - m$).

g defines the rational side, f defines the algebraic side.

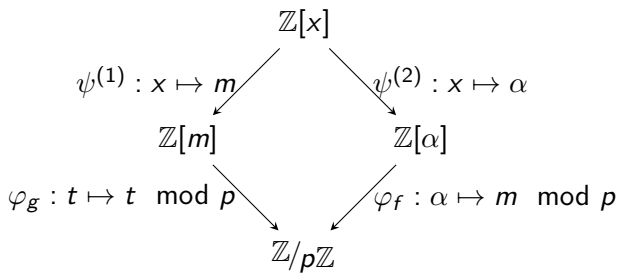
Restating with the resultant

The following restatement can be useful.

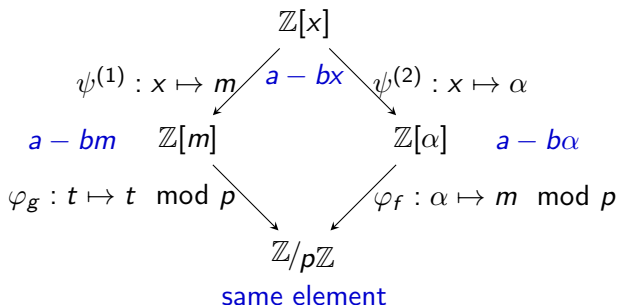
$$f \text{ and } g \text{ share a root modulo } p \Leftrightarrow \text{Res}_x(f, g) = 0 \pmod{p}.$$

Choosing f and g is referred to as the polynomial selection step.

Relations in NFS



Relations in NFS



Take for example $a - bx \in \mathbb{Z}[x]$. Suppose for a moment that:

- the integer $a - bm$ is smooth: product of factor base primes;
- the algebraic integer $a - b\alpha$ is also a product.
- factors on both sides belong to a small set (factor base).

NFS collects many such “good pairs” (a, b) .

Collecting relations

Suppose factor bases are: ● $\{p_1, \dots, p_{99}\}$ (rational),
● $\{\pi_1, \dots, \pi_{99}\}$ (algebraic).

Good pairs could lead to:

$$a_1 - b_1 m = p_2 \times p_4^3 \times p_{12} \times p_{22},$$

$$a_2 - b_2 m = p_1 \times p_3 \times p_5^2 \times p_{47},$$

$$a_3 - b_3 m = p_1^6 \times p_4 \times p_7 \times p_{22},$$

and at the same time:

$$a_1 - b_1 \alpha = \pi_1 \times \pi_3^2 \times \pi_6^2 \times \pi_{35},$$

$$a_2 - b_2 \alpha = \pi_2 \times \pi_8^2 \times \pi_{29},$$

$$a_3 - b_3 \alpha = \pi_2^4 \times \pi_3 \times \pi_{23},$$

This would turn into relations in \mathbb{F}_p :

$$\phi_f(p_1)^6 \times \phi_f(p_4) \times \phi_f(p_7) \times \phi_f(p_{22}) = \phi_g(\pi_2)^4 \times \phi_g(\pi_3) \times \phi_g(\pi_{23}).$$

Caveat

This is too rosy. $\mathbb{Z}[\alpha]$ not a UFD. Complications ahead.

Several obstructions

There are many tiny technicalities to work with:

- We would like to know the **ring of integers**, but we cannot compute it for certain;
- We need to **factor into ideals**. Most often, it's easy.
- Some “bad ideals” need to be dealt with.
For factoring, we can cheat on them, not for DL.
- Because of number field units, we must add ℓ -adic characters called **Schirokauer maps**.
- We need to define **virtual logarithms** properly.

None is a real deterrent to making NFS-DL a **completely practical** algorithm.

NFS summary

Outline of the algorithm:

- Do the setup. Choose a factor base bound B ;
- Relation search
 - Pick pairs a, b for coprime integers a and b ;
 - Expect $a - bm$ to be a smooth integer ;
 - Expect also the ideal $(a - b\alpha)$ to be smooth ;
- Compute Schirokauer maps. Do linear algebra to compute virtual logarithms of factor base elements.
- Compute individual logarithms.

Finding smooth a, b

Finding a, b such that $a - bm$ is smooth: easily stated.

Finding a, b such that $(a - b\alpha)\mathcal{O}_K$ is a smooth ideal:

- When $I = \mathfrak{p}_1^{e_1} \cdots \mathfrak{p}_k^{e_k}$, we have $\text{Norm } I = \prod_i (\text{Norm } \mathfrak{p}_i)^{e_i}$.
- Look at $\text{Norm}((a - b\alpha)\mathcal{O}_K) = \text{Norm}_{K/\mathbb{Q}}(a - b\alpha) = b^{\deg f} f(a/b) (\in \mathbb{Z})$.
- If this norm is smooth, then $(a - b\alpha)\mathcal{O}_K$ is a smooth ideal.

Each pair a, b meeting these conditions yields a [relation](#).

For each relation, we focus on valuations at primes / prime ideals.

Plan

NFS-DL

NFS-DL setting and headaches

Rough analysis

Analysis of NFS

We consider a, b bounded by $L_p[1/3]$, and $B = L_p[1/3]$.

Polynomials f and g can be chosen in various ways. Usual setup:

- $|f_i| \approx L_p[2/3]$ and $\deg f \approx \left(\frac{\log p}{\log \log p}\right)^{1/3} = \log_{\log p} L_p[1/3]$.
- $|g_i| \approx L_p[2/3]$ and $\deg g = 1$.

We neglect the **time for testing smoothness** (see later).

Approximate sizes:

- $a - bm \approx L_p[2/3]$;
- $\text{Norm}(a - b\alpha) = b^{\deg f} f(a/b) \approx L_p[2/3]$.

Smoothness probability is thus $L_p[1/3]$.

\rightsquigarrow relation collection, linear algebra both in $L_p[1/3]$.

With the constants

More precise complexity for NFS-DL in \mathbb{F}_p :

- for **arbitrary** p : $L_p[1/3, (64/9)^{1/3}]$ (same as GNFS);
- for **exceptionally easy** p : $L_p[1/3, (32/9)^{1/3}]$ (same as SNFS).
Example of an easy p : $2^{1120} + 2^{161} - 1$.

Note that this complexity difference lies in the **exponent**.

This makes a huge difference in practice.

With the constants

More precise complexity for NFS-DL in \mathbb{F}_p :

- for **arbitrary** p : $L_p[1/3, (64/9)^{1/3}]$ (same as GNFS);
- for **exceptionally easy** p : $L_p[1/3, (32/9)^{1/3}]$ (same as SNFS).
Example of an easy p : $2^{1120} + 2^{161} - 1$.

Note that this complexity difference lies in the **exponent**.

This makes a huge difference in practice.

A fact which should be better known

Some **exceptionally easy** p conspicuous.

But disguising an easy p as an honest random-looking p is **easy**:

- pick f and g to our liking;
- publish $p = \text{Res}(f, g)$.

Bottom line: never ever let a 3rd party (say, NSA) choose your p .

Plan

History and setting

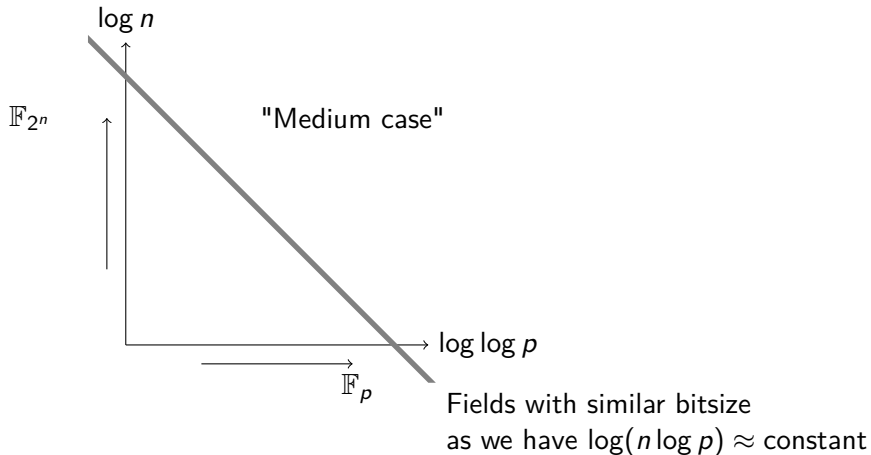
General setup – easy example with FFS

NFS-DL

DLP in \mathbb{F}_{p^n}

$L(1/3)$ for large genus, small degree curves

Finite fields \mathbb{F}_{p^n} w.r.t. p and n



Algorithms for DLP in \mathbb{F}_{p^n}

“Modern” algorithms have improved the complexity somewhat.

The complexity depends much on how n compares to $\log p$.

- DLP in $\mathbb{F}_{p^n}^\times$ for $\log p$ small compared to n : [Cop84], and the Function Field Sieve (FFS) [Adl94, AH99, JL02]:

$$L_q(1/3, 1.53).$$

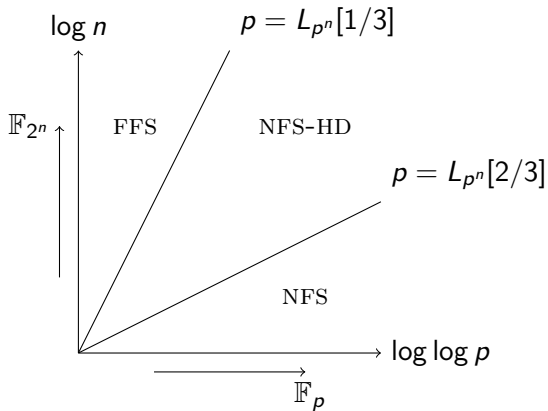
- DLP in $\mathbb{F}_{p^n}^\times$ for n small compared to $\log p$: The Number Field Sieve for DLP (several successive variants [Gor93, Sch99]):

$$L_q(1/3, 1.93).$$

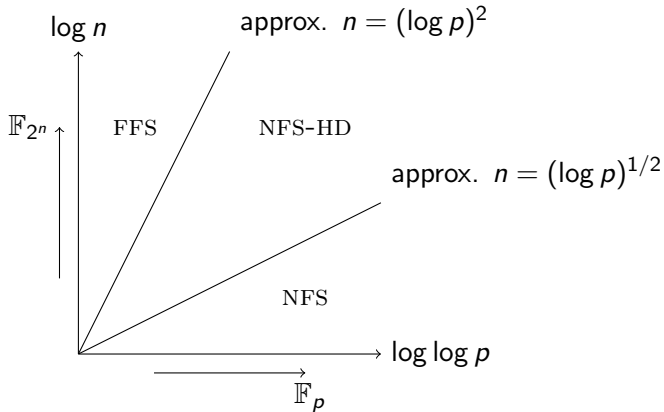
- DLP in $\mathbb{F}_{p^n}^\times$ for fields inbetween: NFS-HD [JLSV06] + some recent improvements in 2014 (Joux, Pierrot):

$$L_q(1/3, c) \quad (\text{for some } c).$$

Situation in 2006



Situation in 2006



Number field sieve for extensions

Until 2006, only $L[1/2]$ algorithms were known to compute DL in \mathbb{F}_{p^n} with $L[1/3] \leq p \leq L[2/3]$.

Wanted: polynomials with a common root in \mathbb{F}_{p^n} .

Hint: in this setting, neither p nor n are extremely big: $p \leq L[1/3]$.

Fairly simple idea:

- Take $f(x)$ of degree n , irreducible modulo p , with small integer coefficients.
- Take $g(x) = f(x) + p$.

All required conditions are met. This gives an $L[1/3]$ algorithm.

More to come

More recent work brings improvements:

- look forward to talk by C. Pierrot;
- look forward to talk by A. Guillevic.

Plan

History and setting

General setup – easy example with FFS

NFS-DL

DLP in \mathbb{F}_{p^n}

$L(1/3)$ for large genus, small degree curves

$L(1/3)$ on large genus curves

Pick a group family similar to the one vulnerable to [ADH94].

- Fixed base field;
- Growing genus.

Additional requirement: we want the curve C_g of genus g with a defining polynomial $F(x, y)$ of **roughly balanced degree**:

$$g^{1/3} \leq \deg_x F \leq g^{2/3},$$
$$g^{1/3} \leq \deg_y F \leq g^{2/3},$$

We might for example imagine a family of $C_{a,b}$ curves, although the $C_{a,b}$ condition is not really a requirement.

Question: do we have an edge on solving the DLP on $\text{Jac}_{C_g}(\mathbb{F}_q)$?

$L(1/3)$ on large genus curves

Among prime divisors on $\text{Jac}_{C_g}(\mathbb{F}_q)$, define **factor base** \mathcal{F} as:

- $\langle u(x), P(x, y) \rangle$;
- $u(x)$ irreducible, of **small degree** $\geq g^{1/3}$;
- Plus finitely many divisors (singularities, places at ∞).

Splitting of functions on the factor base

Let $\phi(x, y) \in \mathbb{F}_q[x, y]$ be an arbitrary function.

- if $\deg_x \text{Res}_y(\phi, F)$ is $g^{1/3}$ -smooth, then $\text{div } \phi$ can be written as a sum of divisors in \mathcal{F} .
- The probability of smoothness is same as in Canfield-Erdős-Pomerance ([Heß04]).
For $\deg_x \text{Res}_y = \log_q L_{q^g}[a, \alpha]$, $\text{Prob} = L_{q^g}[a - 1/3, \cdot]$.

DLP algorithm from smooth functions

The factor base we have defined has size $L_{q^g}[1/3, \cdot]$.

How can we choose ϕ so that $\deg_x \text{Res}_y(\phi, F) \leq g^{2/3}$?

- Simply write down how the degrees in ϕ and F interact:
$$\deg_x \text{Res}_y(\phi, F) = \deg_x \phi \deg_y F + \deg_y \phi \deg_x F.$$
- Provided $\deg_x F$ and $\deg_y F$ stay within $[g^{1/3}, g^{2/3}]$, we can achieve that.
 - e.g. $(1/2, 1/2) \rightsquigarrow \deg_x \phi = \deg_y \phi = g^{1/6}$.
 - e.g. $(3/8, 5/8) \rightsquigarrow \deg_x \phi = g^{1/24}, \deg_y \phi = g^{7/24}$.

Bottom line

We get an $L(1/3)$ algorithms for virtual logs of elements of \mathcal{F} .

The individual logs step is tricky, but works too.

Part 3

Computationally hard tasks in NFS

Sieving

Linear algebra

NFS-DL/FFS Guinness book

Plan

Sieving

Linear algebra

NFS-DL/FFS Guinness book

Plan

Sieving

- Sieving basics

- Lattice sieving

Sieving

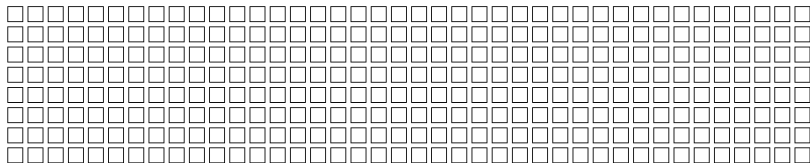
A possible way to check $(a - b\alpha)$ for smoothness:

- Consider all (a, b) in the search area, one after another.
 - Either factor $\text{Norm}(a - b\alpha)$, and see if it is smooth;
 - Or iterate through all relevant prime ideals \mathfrak{p} instead, and see if they divide. Succeed if the total contribution matches the norm.

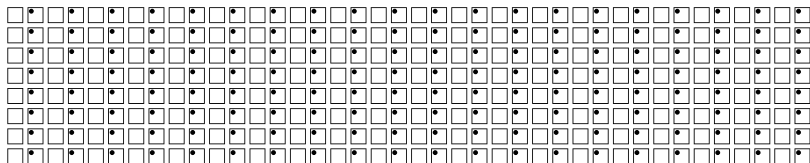
Much better: sieving

- Decide beforehand of a **sieving space**: (a, b) within some area.
- “for each a, b , for each \mathfrak{p} , do” becomes “for each \mathfrak{p} , for each a, b , do”.

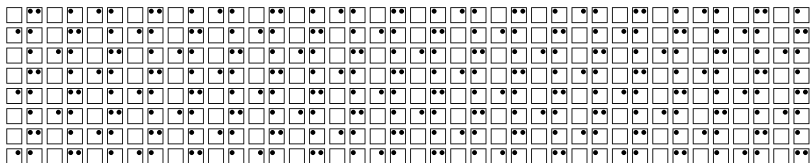
Sieving, visually



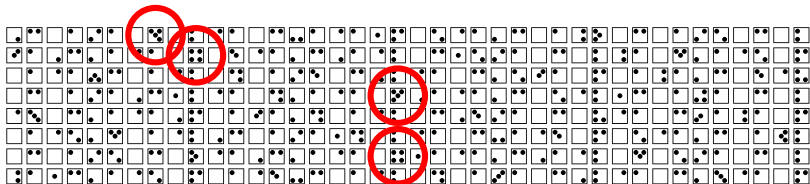
Sieving, visually



Sieving, visually



Sieving, visually



Sieving challenges

How can we parallelize sieving ?

- Imagine an immense sieve area \mathcal{S} (e.g. total size 2^{50} or more).
- We may split \mathcal{S} in many zones of manageable size.
- Problem: the **yield** varies much. Need serious sampling across \mathcal{S} to assess the expected total yield reasonably.
- Diminishing returns.

Nontrivial computations in sieving:

- For each “line” (fixed b) in \mathcal{S} , the computation of the first a to sieve in the line costs a reduction mod p .

Lattice sieving to the rescue.

Old idea (1993), but superiority demonstrated only after 2000.

Plan

Sieving

Sieving basics

Lattice sieving

Lattice sieving

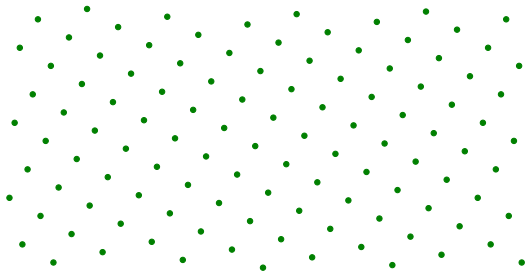
Lattice sieving [Pol93] improves the special- q idea [DH84].

Consider q just above the factor base.

Only a few $(\frac{1}{q})$ points $(a, b) \in \mathcal{S}$ satisfy $q \mid (a - b\alpha)$.

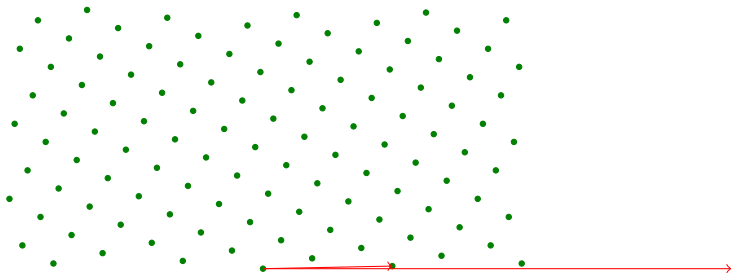
- Sieve **only** these points.
- A lattice structure: $(a, b) = i(a_0, b_0) + j(a_1, b_1)$.

The set of (a, b) points to sieve



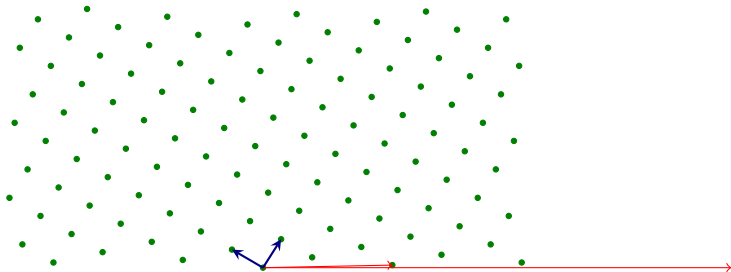
The set of (a, b) points to sieve

There's potential for being stupid in describing the good (a, b) 's.



The set of (a, b) points to sieve

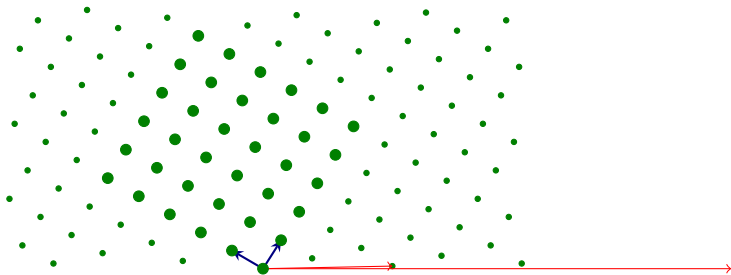
There's potential for being stupid in describing the good (a, b) 's.



- It is much better to use a **reduced basis**.
- Enumeration within $[-S_a, S_a] \times [1, S_b]$ is cumbersome.

The set of (a, b) points to sieve

There's potential for being stupid in describing the good (a, b) 's.



- It is much better to use a **reduced basis**.
- Enumeration within $[-S_a, S_a] \times [1, S_b]$ is cumbersome.
- Consider a **fixed (i, j) plane** instead.

Lattice sieving benefits

- Forced factor in the norm improves the smoothness probability.
- Reduced lattice basis limits the expense of this constraint on the size of (a, b) .
- Sieved areas are smaller, and each is typically manageable on a single node.
- Special- q provide an easy division of tasks, with (almost) stable yield.

More implementation tricks:

- Franke-Kleinjung “sieving by vectors” [FK05].
- Bucket sieving.

Plan

Sieving

Linear algebra

NFS-DL/FFS Guinness book

Sparse linear algebra

Let M be an $N \times N$ matrix over a finite field K . We want to find:

$$w \in K^N \text{ s.t. } Mw = 0.$$

- Factoring or DL: M is **sparse**: $O(\log^2 N)$ non-zeroes per row.
- Factoring: $K = \mathbb{F}_2$; DL over \mathbb{F}_p : $K = \mathbb{F}_\ell$ with $\ell \mid (p - 1)$.
- Space complexity for storing M : $O(N \log^2 N)$.

Linear system solving:

- Gauss: time $O(N^3)$, space $O(N^2)$.
- Recursive, using matrix multiply: time $O(N^\omega)$, space $O(N^2)$.
- None of the options above exploit **sparsity**.

Sparse linear algebra

For matrices arising from crypto contexts, fill-in **cannot** be tolerated.

Some figures from RSA-768

- 192 796 550 rows/columns ;
- 27 797 115 920 non-zero coefficients.
- 105 gigabytes as a sparse matrix.
- >4000 terabytes as a dense bit matrix.

The matrix **cannot be modified** in the course of the computation. We may only use **black-box algorithms**. No access to M itself.

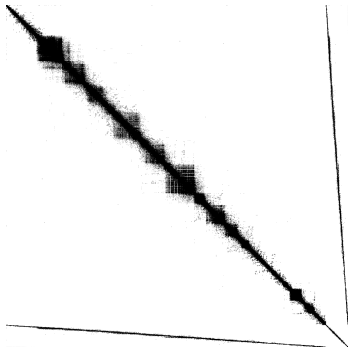
$$v \longrightarrow \boxed{M} \longrightarrow M \times v$$

Comparison with numerical world

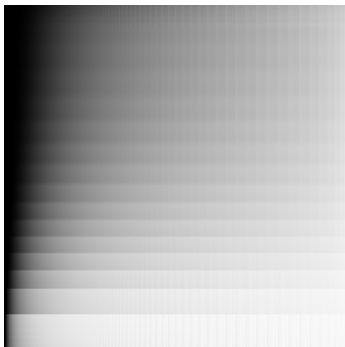
Exact linear algebra differs much from linear algebra over \mathbb{C} .

- No notion of **approximate** solution.
- No notion of **convergence**.

The **matrices** are not the same either:



(some PDE example)



(a factoring matrix)

Linear algebra algorithms

Algorithms used for exact black-box linear algebra.

- Wiedemann algorithm;
- Lanczos algorithm;
- Block Wiedemann algorithm;
- Block Lanczos algorithm;

Which one is preferred depends on the context.

- Lanczos needs fewer matrix operations;
- Wiedemann offers more distribution opportunities.

Plan

Sieving

Linear algebra

NFS-DL/FFS Guinness book

Records in \mathbb{F}_p

- Early NFS-DL works: Weber [Web95];
- Long stream of achievement by Joux and Lercier, following [JL03]: 110dd [JL01a], 120dd [JL01b], 130dd [JL05a].
- More "do like factoring" approach: Kleinjung: 160dd [Kle07].
- Bouvier, Gaudry, Imbert, Jeljeli, T.: 180dd, 2014.

Records over binary fields

Records usually announced on the NMBRTHRY mailing list.

- Old records with Coppersmith's method, which is a special version of the Function Field Sieve: T.: 607 bits [Tho02].
- FFS records by Joux and Lercier (considerably more efficient than Coppersmith's algorithm in this range): 521 bits [JL01c], 613 bits [JL05b].
- Fall 2012: 619 bits [BBD⁺12] (roughly a day of work).
 - About 160 core-hours of [sieving](#).
 - Linear algebra (18hrs) using [graphics cards](#).
- Spring 2013: 809 bits [BBD⁺14].
- And then came the quasi-polynomial variants (forthcoming part):
 - Record for \mathbb{F}_{2^p} : 1279 bits [?Kleinjung14].
 - Record for \mathbb{F}_{2^n} : 9234 bits ($9234 = 2 \cdot 3^5 \cdot 19$) [GKZ14].

Records in the medium range

The **medium case** has also been covered by Joux and Lercier.

- $\mathbb{F}_{65537^{25}}$ ($\sim 120\text{dd}$) [JL05c].
- $\mathbb{F}_{370801^{30}}$ ($\sim 168\text{dd}$) [JL05d].
- There, some tricks related to **Galois action** can be used.

Available software

Many people don't release their code.

Everything done in Nancy is with the CADO-NFS software.

- Originally a factoring software;
- Use for DL is more difficult, but there's some documentation.
- CADO-NFS has been used in the recent FREAK and LOGJAM attacks.

`http://cado-nfs.gforge.inria.fr/`

Part 4

Quasi-polynomial DLP in small characteristic

Instances of the FFS setting

New construction

Descent techniques

Complexity

Plan

Instances of the FFS setting

New construction

Descent techniques

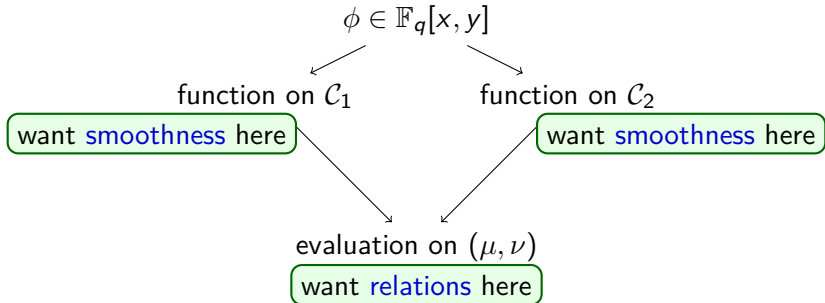
Complexity

The Function Field Sieve

FFS: two plane curves defined over \mathbb{F}_q :

$$\mathcal{C}_1 : C_1(x, y) = 0; \quad \mathcal{C}_2 : C_2(x, y) = 0$$

- such that $\mathcal{C}_1 \cap \mathcal{C}_2$ has a point (μ, ν) defining \mathbb{F}_{q^n} ;
- i.e. $\text{Res}(C_1, C_2)$ has an irreducible factor of degree n .



Examples already seen

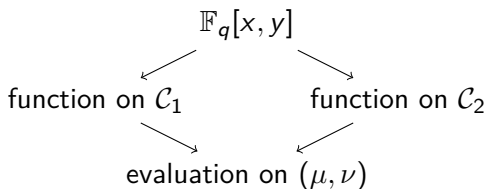
The “easy FFS” scenario:

- $C_1(x, y) = y - f(x)$; $C_2(x, y) = x - g(y)$
with f, g such that $g(f(x)) - x$ has a degree n irreducible factor.
- Both have genus 0, function field is $\mathbb{F}_q(t)$.

The “usual FFS” is more similar to NFS:

- $C_1(x, y) = y - f(x)$; “rational side”;
 $C_2(x, y)$ with degree d in y . “algebraic side”
- The curve C_2 has (very) large genus here, but that is not something to worry much about. Most happens in $\mathbb{F}_q(t)$ anyway.

Back to the general setting



- We have several constructions which fit in this picture;
- For fun, we can even imagine a few more (rarely practical);
- Algebraic technicalities are no obstacle;
- Implementation made efficient with [sieving](#), taking e.g. functions $\phi(x, y) = a(x) - yb(x)$.

Complexity

All FFS-like constructions presented have complexity $L(1/3)$.

- Key: degree of norms on both side which we test for smoothness.
- All constructions have norms of degree $\approx n^{2/3}$.
- Factor bases bounds are chosen as $\beta \approx n^{1/3}$.
 - Smaller β : smoothness too rare;
 - Larger β : unwieldy linear system.

Plan

Instances of the FFS setting

New construction

Descent techniques

Complexity

New construction

Guiding idea:

- choose \mathcal{C}_2 with **built-in smooth decompositions** for **small degree functions** ϕ .
- use the FFS setting flexibility to make this work for \mathbb{F}_{q^n} .

New construction

Guiding idea:

- choose \mathcal{C}_2 with **built-in smooth decompositions** for **small degree functions** ϕ .
- use the FFS setting flexibility to make this work for \mathbb{F}_{q^n} .

How do we choose \mathcal{C}_2 ?

Pick $\mathcal{C}_2 = y^q - x$ and $\phi = y - x$. (original variant: $y - x^q$)

- As a function on \mathcal{C}_2 , we have $\phi \equiv y^q - y = \prod_{\alpha \in \mathbb{F}_q} (y - \alpha)$.
- Nice, but this makes only **one relation**.
- Can we expand it into more relations ?
 - Write similar relations;
 - Write down set of ϕ which lead to them.

Relations which look alike

Joux considered replacing y by $\frac{ay+b}{cy+d}$.

$$\left(\frac{ay+b}{cy+d}\right)^q - \frac{ay+b}{cy+d} = \prod \left(\frac{ay+b}{cy+d} - \alpha\right).$$

- If $a, b, c, d \in \mathbb{F}_q$ this brings nothing interesting.
- We may however **base extend** to \mathbb{F}_{q^k} for some $k > 1$.

$$\phi_{a,b,c,d} = (a^q x + b^q)(cy + d) - (ay + b)(c^q x + d^q)$$

All functions created this way will be smooth on \mathcal{C}_2 by construction.

They have small degree in x and y . Promising.

How many functions ?

$$\phi_{a,b,c,d} = (a^q x + b^q)(cy + d) - (ay + b)(c^q x + d^q)$$

We have as many functions ϕ as we find ways $\text{div } \phi$ may split.

- $y - x$ has q zeroes: $(\alpha : \alpha : 1)$ and a pole: $(1 : 0 : 0) = (\infty)$.
- $\text{div}(y - x) + (q + 1)(\infty)$ is effective of weight $(q + 1)$.
- Same for $\text{div } \phi_{a,b,c,d} + (q + 1)(\infty)$.
- Its support is the **image set of $\mathbb{P}^1(\mathbb{F}_q)$ within $\mathbb{P}^1(\mathbb{F}_{q^k})$** under the homography $y \mapsto \frac{ay+b}{cy+d}$.

We have $\# \text{PGL}_2(\mathbb{F}_{q^k})$ homographies;

Cosets modulo $\text{PGL}_2(\mathbb{F}_q)$ yield identical image sets.

For $k = 2$, this gives **$q^3 + q$ functions to choose from.**

$$\begin{aligned} & q^{3k} - q^k \\ & \div q^3 - q. \end{aligned}$$

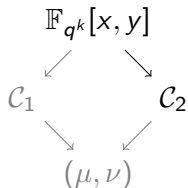
Not quite there yet

- We extended the base field to \mathbb{F}_{q^k} .
- Set of ϕ with $\deg_x = \deg_y = 1$.

$$\mathbb{F}_{q^k}[x, y] \begin{array}{c} \searrow \\ \mathcal{C}_2 \end{array}$$

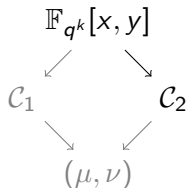
Not quite there yet

- We extended the base field to \mathbb{F}_{q^k} .
- Set of ϕ with $\deg_x = \deg_y = 1$.
- Still need to find \mathcal{C}_1 .
- Our target is \mathbb{F}_{q^n} , or some overfield.



Not quite there yet

- We extended the base field to \mathbb{F}_{q^k} .
- Set of ϕ with $\deg_x = \deg_y = 1$.
- Still need to find \mathcal{C}_1 .
- Our target is \mathbb{F}_{q^n} , or some overfield.



Take \mathcal{C}_1 fairly simple too

$$\mathcal{C}_1 = h_1(x)y - h_0(x).$$

If $h_1(t^q)t - h_0(t^q)$ has a degree n irreducible factor, bingo.

Requires some constraints be met:

- $\deg h_i = \delta \geq 2$ (constant);
- coefficients of h_i in \mathbb{F}_{q^k} ;
- and of course: $n \leq \delta q + 1$.

The constraint on q

What does the constraint on q imply if we target, say, $\mathbb{F}_{2^{1279}}$?

- We are ready to accept working in a mildly larger field;
- Taking $\delta = 2$ would call for \mathbb{F}_q to be at least $\mathbb{F}_{2^{10}}$.
- With $\delta = 5$, $q = 256$ suffices.
- We're bound to need another base extension.

The constraint on q

What does the constraint on q imply if we target, say, $\mathbb{F}_{2^{1279}}$?

- We are ready to accept working in a mildly larger field;
- Taking $\delta = 2$ would call for \mathbb{F}_q to be at least $\mathbb{F}_{2^{10}}$.
- With $\delta = 5$, $q = 256$ suffices.
- We're bound to need another base extension.

Base extension summary

Overall, for solving DLP in \mathbb{F}_{p^n} , we consider instead

$$\mathbb{F}_{q^{kn}} = \mathbb{F}_{p^{mn}}, \text{ with } m = O(\log n).$$

Eventually, because of base extensions, our difficulty measure n becomes $O(n \log n)$. No big deal.

Note: for composite n , the required subfields might be built-in.

Setting summary

- Base field is \mathbb{F}_{q^k} , with q large enough and $k \geq 2$.
- Two simple genus 0 curves:

$$\mathcal{C}_1 : h_1(x)y - h_0(x) = 0,$$

$$\mathcal{C}_2 : y^q = x.$$

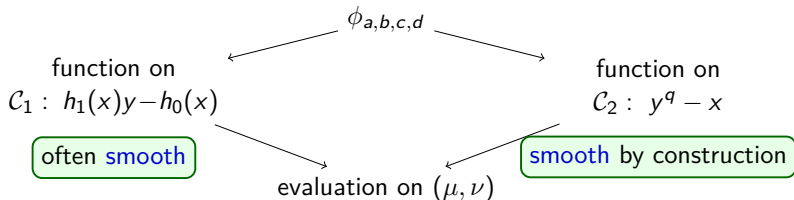
- Pick $\phi_{a,b,c,d} = (a^q x + b^q)(cy + d) - (ay + b)(c^q x + d^q)$;

On the \mathcal{C}_2 side, we split into degree one polynomials in y .

On the \mathcal{C}_1 side, $\text{Res}(\phi, \mathcal{C}_1)$ has **degree $\delta + 1$ in x** .

- For δ a constant, we have a constant proportion of ϕ 's which are smooth on the \mathcal{C}_1 side.
- For $k = 2$: linear system of size $\Theta(q^3 + q) \times (q^2 + 1)$.
- **Heuristically** full rank.

Setting summary



What is the **factor base**, exactly ?

Our mechanism reaches smoothness with **degree 1 factors**.

- relations are between $\{\mu - a\}$ and $\{\nu - b\}$ for $a, b \in \mathbb{F}_{q^k}$
- factor base folding: $\{\mu - a\} = \{(\nu - b)^q\}$.

Complexity: $O(q^{\min(kw, 2k+1)}) = \text{polynomial time}$.

This only gives us degree 1 objects.

Relation collection is now easy

What have we achieved ?

- We have **new polynomial-time algorithm** which is able to **collect relations** and **find logarithms** for elements of degree 1.
- This is something we did not have before. Even the log of something “small” was hard to guess.
- We need to work on a **descent procedure** which makes it possible to relate arbitrary log's to known ones.

We have deliberately omitted some technicalities (“traps”) which change nothing to the general picture.

Plan

Instances of the FFS setting

New construction

Descent techniques

Complexity

Problem statement

Input: an element $P(\mu) \in \mathbb{F}_{q^{kn}}$, for $\deg P > 1$.

Output: a rewritten expression:

$$\log_g P(\mu) = \sum_{a \in \mathbb{F}_{q^k}} e_a \log_g(\mu - a).$$

The descent

Assumption: we have precomputed log's of factor base elements.
The **descent** is the art of turning the DLP computation into a recursive problem.

One descent step

Input: a challenge discrete logarithm to compute for an element of some given size.

Output: an identity which allows to derive the desired log from log's of smaller elements.








Historical notes:

- Coppersmith84 was the first algorithm featuring a descent.
- Many of the NFS-DL/FFS papers from 1993 to 2006 introduced new (not always correct) variations for the descent.

Example

Start with $h =$  (some integer).

- Start with a relation involving h :








$$\begin{aligned} a - bm &= \text{} \times \text{} \times \cdots \times \text{} \times \text{,} \\ (a - b\alpha) &= \text{} \times \cdots \times \text{} \times \text{.$$

- Then find relations killing all the outstanding terms:





















Example

Start with $h =$  (some integer).

- Start with a relation involving h :

$$\begin{aligned} a - bm &= \text{} \times \text{} \times \cdots \times \text{} \times \text{,} \\ (a - b\alpha) &= \text{} \times \cdots \times \text{} \times \text{}.\end{aligned}$$

- Then find relations killing all the outstanding terms:

$$\begin{aligned} a - bm &= \text{} \times \text{} \times \cdots \times \text{}; & (a - b\alpha) &= \text{} \times \cdots \times \text{}; \\ a - bm &= \text{} \times \cdots \times \text{}; & (a - b\alpha) &= \text{} \times \text{} \times \cdots \times \text{}; \\ a - bm &= \text{} \times \text{} \times \cdots \times \text{}; & (a - b\alpha) &= \text{} \times \cdots \times \text{}; \\ a - bm &= \text{} \times \cdots \times \text{}; & (a - b\alpha) &= \text{} \times \text{} \times \cdots \times \text{};\end{aligned}$$

Descent tools

Several **recursive techniques** have been developed.

- Same techniques as for FFS are still valid;
- Polynomial system based techniques; [Joux13]
- General QPA elimination; [BGJT14]
- On the fly elimination for even degree P . [GKZ14]

In practice, all techniques may be combined.

Descent tools

Wanted: ϕ with $P(\mu)$ involved in the corresponding relation.

- Strategy 1: force $P(x)$ to appear on the \mathcal{C}_1 side.
- Strategy 2: force $\check{P}(y)$ to appear on the \mathcal{C}_2 side.
(with $\check{P}(y) = \sum_i p_i^{q^{k-1}}(y)$, such that $(\check{P}(y))^q = P(x)$)

In all cases, we want the relation obtained to have $P(\mu)$ as the **largest degree thing**, with other terms having smaller degree.

Strategy 1 is in the polynomial-system-based approach, as well as the on-the-fly even degree elimination.

Strategy 2 is in the QPA article.

Forcing \check{P} on the \mathcal{C}_2 side

Start with the easy choice: $\phi = P(x) - \check{P}(y)$.

- On \mathcal{C}_2 , this rewrites as

$$(\check{P}(y))^q - \check{P}(y) = \prod_{\alpha \in \mathbb{F}_q} \check{P}(y) - \alpha.$$

- This is **one relation**, but:
 - We are not sure this ϕ gives smooth \mathcal{C}_1 side.
 - The relation involves many other polynomials of same degree.
- Use homographies again: let $\frac{a\check{P}+b}{c\check{P}+d}$ appear.

$$\phi_{a,b,c,d}^{(P)} = (a^q P(x) + b^q)(c\check{P}(y) + d) - (a\check{P}(y) + b)(c^q P(x) + d^q)$$

Forcing \check{P} on the \mathcal{C}_2 side

$$\phi_{a,b,c,d}^{(P)} = (a^q P(x) + b^q)(c\check{P}(y) + d) - (a\check{P}(y) + b)(c^q P(x) + d^q)$$

- Same machinery; a, b, c, d run over $\mathrm{PGL}_2(\mathbb{F}_{q^k})/\mathrm{PGL}_2(\mathbb{F}_q)$.
- On side \mathcal{C}_1 , we have a polynomial of degree $(\delta + 1) \deg P$.
- A constant proportion of the a, b, c, d yield smooth \mathcal{C}_1 side. For example we may target $\frac{1}{2} \deg P$ smoothness.

Halfway there

We have $\Theta(q^3 + q)$ (for $k = 2$) relations of the kind $\mathcal{L}_i = \mathcal{R}_i$.

- On the left, we have factors f with $\deg f \leq \frac{1}{2} \deg P$ (e.g.).
- On the right, factors among $\{P - \gamma, \gamma \in \mathbb{F}_{q^k}\}$.

Letting only P show up

Our starting polynomial P sometimes appears in \mathcal{R}_i .

We now do a **multiplicative combination** of all relations.

$$\prod_i \mathcal{L}_i^{e_i} \equiv \prod_i \mathcal{R}_i^{e_i}.$$

- We would like to have $\prod_i \mathcal{R}_i^{e_i} \equiv P$.
- This is a linear system. We need the vector $(1, \overbrace{0, \dots, 0}^{q^2})$ be in the image (assuming some adequate indexing).

Can we solve the linear system ?

For $\prod_i \mathcal{R}_i^{e_i} \equiv P$ to be possible, we need:

Heuristic

The linear system of size $\Theta(q^3 + q) \times (q^2 + 1)$ has full rank.

Note: if we hadn't restricted our view to smooth relations:

- We would have a system of size $(q^3 + q) \times (q^2 + 1)$.
- Combinatorial design theory recognizes an [inversive plane](#) there. This proves that the (big) matrix has full rank.
- Yet, saying something about our few selected rows is difficult.

Which kind of relation do we get ?

For splitting **one** polynomial P , we need a multiplicative combination of as many as $q^2 + 1$ left-hand sides \mathcal{L}_i !

This means that the **arity of the descent tree** will be large:

- $O(q^2)$ LHS expressions.
- $O(\deg P)$ terms per LHS.
- Arity at most $O(q^2 \deg P)$.

Other descent techniques

First descent method found by Joux forces P on the \mathcal{C}_1 side:

$$\text{take } \phi = u_1(x)\check{u}_2(y) - \check{u}_1(y)u_2(x)$$

- Put indeterminates for coefficients of u_1 and u_2 .
- The condition “ P appears on the \mathcal{C}_1 side” is a **bilinear polynomial system**.
- To get enough degrees of freedom, we need:

$$\deg u_1 + \deg u_2 > \deg P.$$

Bilinear polynomial system solving: $\min(\deg u_1, \deg u_2)$ matters.

- Any solution to the system will do.
- We get arity $O(q)$ at each step.
- Solving the system is not completely cheap though.
- Not enough to reach quasi-polynomial complexity.

On-the-fly technique by Granger et al.

Assume P has degree $2d$. Note that P splits in $\mathbb{F}_{q^{kd}}$.

$$P(x) = f(x) \times f^\sigma(x) \times \cdots \times f^{\sigma^{d-1}}(x),$$

where all f_i are quadratic polynomials over $\mathbb{F}_{q^{kd}}$.

- Base-extend to $\mathbb{F}_{q^{kd}}$ and find appropriate rewriting for $f(x)$ into linear polynomials;
Beware: $\mathbb{F}_{q^{kd}}$ is large !
- Take $\text{Norm}_{\mathbb{F}_{q^{kd}}/\mathbb{F}_{q^k}}$ to obtain a rewritten expression for P .
 - P is the only term of degree $2d$;
 - All other terms have degree dividing d .
- Arity obtained is $O(q)$.
- Extreme case: degrees which are powers of two are best !

Plan

Instances of the FFS setting

New construction

Descent techniques

Complexity

The descent tree

Each node of the descent tree corresponds to one application of the new descent tool, hence its arity is in $O(q^2 \deg P)$, or $O(q)$ for $\deg P$ even.

level	$\deg P_i$	breadth of tree
0	D	1
1	$D/2$	$q^2 D$
2	$D/4$	$q^2 D \cdot q^2 \frac{D}{2}$
3	$D/8$	$q^2 D \cdot q^2 \frac{D}{2} \cdot q^2 \frac{D}{4}$
\vdots	\vdots	\vdots
$\log D$	1	$\leq q^{2 \log D} D^{\log D}$

Total number of nodes = $q^{O(\log D)}$.

Each node entails a cost which is polynomial in q .

Bottom of the tree

At level $\log D$, we are below some constant degree (say 2).

Several techniques may be used for descending from there to the set of known logs:

- Continue with the descent procedure so as to get something overdetermined;
- Use the ad hoc degree 2 technique from [GGMZ13,GKZ14].

Overall complexity

$q^{O(\log D)}$ nodes, $O(q^5)$ per node \rightsquigarrow Complexity $q^{O(\log D)}$.

Main result

Main result

Discrete logarithms in \mathbb{F}_{q^n} can be computed in heuristic time

$$\max(q, n)^{O(\log n)}.$$

- Example for $K = \mathbb{F}_{2^p}$:
- Base-extend to \mathbb{F}_q with $q \approx p$.
 - Cost is $p^{O(\log p)} = \exp(O((\log p)^2))$.
 - This is **quasi-polynomial**.

Conclusion

- New algorithm
- Quasi-polynomial complexity.
 - Relies on heuristic, experimentally checked.

Cross-over with existing techniques ?

- Some contexts are **totally unsafe** now:
 - Highly composite extension degrees (many subfields);
 - Fields from small characteristic pairings.
- Striking computational records for these easy targets.
- More general case: [GKZ14] has clearly made **FFS obsolete**.
 - Tackled $\mathbb{F}_{2^{1279}}$;
 - The on-the-fly technique is a clear asset.
 - Combination of tools used.

Take home:

- RIP small characteristic pairings.
- No, this does **not** affect RSA, nor large characteristic DLP.

References I

- [Adl79] L. M. Adleman, *A subexponential algorithm for the discrete logarithm problem with applications to cryptography*, 20th Annual Symposium on Foundations of Computer Science (FOCS '79), 1979, pp. 55–60. San Juan, Puerto Rico, October 29–31, 1979.
- [Adl94] ———, *The function field sieve*, ANTS-I, 1994, pp. 108–121. 1st Algorithmic Number Theory Symposium, Cornell University, May 6–9, 1994.
- [ADH94] L. M. Adleman, J. DeMarras, and M.-D. Huang, *A subexponential algorithm for discrete logarithms over the rational subgroup of the jacobians of large genus hyperelliptic curves over finite fields*, ANTS-I, 1994, pp. 28–40. 1st Algorithmic Number Theory Symposium, Cornell University, May 6–9, 1994.
- [AH99] L. M. Adleman and M.-D. Huang, *Function field sieve methods for discrete logarithms over finite fields*, Inform. and Comput. **151** (1999), no. 1, 5–16.

References II

- [BBD⁺12] Răzvan Bărbulescu, Cyril Bouvier, Jérémie Detrey, Pierrick Gaudry, Hamza Jeljeli, Emmanuel Thomé, Marion Videau, and Paul Zimmermann, *The relationship between some guy and cryptography*, 2012. ECC2012 rump session talk (humoristic), short computation report.
- [BBD⁺14] ———, *Discrete logarithms in $\text{GF}(2^{809})$ with FFS*, Public Key Cryptography - PKC 2014, 2014, pp. 221–238. Proc. 17th International Conference on Practice and Theory in Public Key Cryptography, Buenos Aires, Argentina, Mar. 26-28, 2014.
- [BF01] D. Boneh and M. Franklin, *Identity-based encryption from the Weil pairing*, Advances in Cryptology – CRYPTO 2001, 2001, pp. 213–229. Proc. 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001.
- [BLS04] D. Boneh, B. Lynn, and H. Shacham, *Short signatures from the Weil pairing*, J. Cryptology **17** (2004), no. 4, 297–319.

References III

- [Cop84] D. Coppersmith, *Fast evaluation of logarithms in fields of characteristic two*, IEEE Trans. Inform. Theory **IT-30** (1984), no. 4, 587–594.
- [DH84] J. A. Davis and D. B. Holridge, *Factorization using the quadratic sieve algorithm*, Advances in Cryptology – CRYPTO '83, 1984, pp. 103–113. Proc. Cryptology Workshop, Santa Barbara, CA, August 22–24, 1983.
- [EGT11] Andreas Enge, Pierrick Gaudry, and Emmanuel Thomé, *An $L(1/3)$ discrete logarithm algorithm for low degree curves*, J. Cryptology **24** (2011), no. 1, 24–41.
- [FK05] Jens Franke and Thorsten Kleinjung, *Continued fractions and lattice sieving*, Special-purpose hardware for attacking cryptographic Systems – SHARCS, 2005.
- [Gor93] D. M. Gordon, *Discrete logarithms in $\text{GF}(p)$ using the number field sieve*, SIAM J. Discrete Math. **6** (1993), no. 1, 124–138.

References IV

- [GKZ14] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel, *Discrete Logarithms in $GF(2^{234})$* , January 2014. Email to the NMBRTHRY mailing-list.
- [Heß04] F. Heß, *Computing relations in divisor class groups of algebraic curves over finite fields*, 2004. Preprint, submitted to J. Symbolic Comput. Available at <http://www.staff.uni-oldenburg.de/florian.hess/publications/dlog.pdf>.
- [Jou00] Antoine Joux, *A one round protocol for tripartite Diffie-Hellman*, ANTS-IV, 2000, pp. 385–393. 4th Algorithmic Number Theory Symposium, Leiden, The Netherlands, July 2–7, 2000.
- [JL01a] Antoine Joux and Reynald Lercier, *Discrete logarithms in $GF(p)$ (110 decimal digits)*, January 2001. Email to the NMBRTHRY mailing-list.
- [JL01b] _____, *Discrete logarithms in $GF(p)$ (120 decimal digits)*, April 2001. Email to the NMBRTHRY mailing-list.
- [JL01c] _____, *Discrete logarithms in $GF(2^n)$ (521 bits)*, September 2001. Email to the NMBRTHRY mailing-list.

References V

- [JL02] ———, *The function field sieve is quite special*, ANTS-V, 2002, pp. 431–445. 5th Algorithmic Number Theory Symposium, Sydney, Australia, July 2002.
- [JL03] ———, *Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the gaussian integer method*, Math. Comp. **72** (2003), no. 242, 953–967.
- [JL05a] ———, *Discrete logarithms in $\text{GF}(p)$ - 130 digits*, June 2005. Email to the NMBRTHRY mailing-list.
- [JL05b] ———, *Discrete logarithms in $\text{GF}(2^{607})$ and $\text{GF}(2^{613})$* , September 2005. Email to the NMBRTHRY mailing-list.
- [JL05c] ———, *Discrete logarithms in $\text{GF}(65537^{25})$ - 120 digits - 400 bits*, October 2005. Email to the NMBRTHRY mailing-list.
- [JL05d] ———, *Discrete logarithms in $\text{GF}(370801^{30})$ - 168 digits - 556 bits*, November 2005. Email to the NMBRTHRY mailing-list.

References VI

- [JLSV06] Antoine Joux, Reynald Lercier, Nigel P. Smart, and Frederik Vercauteren, *The number field sieve in the medium prime case*, Advances in Cryptology – CRYPTO 2006, 2006, pp. 326–344. Proc. 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20–24, 2006.
- [Kle07] Thorsten Kleinjung, *Discrete logarithms in $GF(p)$ - 160 digits*, May 2007. Email to the NMBRTHRY mailing-list.
- [LL93] Arjen K. Lenstra and H. W. Lenstra Jr (eds.), *The development of the number field sieve*, Lecture Notes in Math., vol. 1554, Springer–Verlag, 1993.
- [LV00] Arjen K. Lenstra and E. R. Verheul, *The XTR public key system*, Advances in Cryptology – CRYPTO 2000, 2000, pp. 1–19. Proc. 20th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2000.

References VII

- [Mau94] U. M. Maurer, *Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms*, Advances in Cryptology – CRYPTO '94, 1994, pp. 271–281. Proc. 14th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 21–25, 1994.
- [MW96] U. M. Maurer and S. Wolf, *Diffie-Hellman oracles*, Advances in Cryptology – CRYPTO '96, 1996, pp. 268–282. Proc. 16th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 1996.
- [MSV04] A. Muzereau, Nigel P. Smart, and Frederik Vercauteren, *The equivalence between the DHP and DLP for elliptic curves used in practical applications*, LMS J. Comput. Math. **7** (2004), 50–72.
- [Nec94] V. I. Nechaev, *Complexity of a determinate algorithm for the discrete logarithm*, Mathematical Notes **55** (1994), no. 2, 165–172.
- [vOW99] P. C. van Oorschot and M. J. Wiener, *Parallel collision search with cryptanalytic applications*, J. Cryptology **12** (1999), 1–28.

References VIII

- [Pol93] J. M. Pollard, *The lattice sieve*, The development of the number field sieve, 1993, pp. 43–49.
- [QD90] Jean-Jacques Quisquater and J.-P. Delescaille, *How easy is collision search? Application to DES*, Advances in Cryptology – EUROCRYPT '89, 1990, pp. 429–434. Proc. Eurocrypt '89, Houthalen, April 10–13, 1989.
- [RS08] Karl Rubin and Alice Silverberg, *Compression in Finite Fields and Torus-Based Cryptography*, SIAM J. Comput. **37** (2008), no. 5, 1401–1428.
- [Sch99] O. Schirokauer, *Using number fields to compute logarithms in finite fields*, Math. Comp. **69** (1999), no. 231, 1267–1283.
- [Sho97] Victor Shoup, *Lower bounds for discrete logarithms and related problems*, Advances in Cryptology – EUROCRYPT '97, 1997, pp. 256–266. Proc. International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 1997.

References IX

- [Tho02] Emmanuel Thomé, *Discrete logarithms in $\text{GF}(2^{607})$* , 2002/02/23. Email to the NMBRTHRY mailing-list, short computation report.
- [Web95] D. Weber, *An implementation of the general number field sieve to compute discrete logarithms mod p* , Advances in Cryptology – EUROCRYPT '95, 1995, pp. 95–105. Proc. International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 1995.