A kilobit hidden SNFS discrete logarithm computation

ia.cr/2016/961 (Eurocrypt 2017)

J. Fried[1], P. Gaudry[2], N. Heninger[1], E. Thomé[2]
[1]U. Penn; [2]Caramba/Inria/Loria

Nov 13rd, 2017

Plan

$(\mathbb{Z}/p\mathbb{Z})^*$ in crypto

Backdooring primes

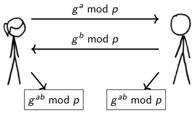
Can one unveil the trapdoor?

Computing DL mod 1024-bit primes with Cado-NFS

Outcome and lessons

$(\mathbb{Z}/p\mathbb{Z})^*$, a.k.a. MODP groups

For Diffie-Hellman, for DSA: we've been using $(\mathbb{Z}/p\mathbb{Z})^*$ groups for decades.



Today (and whether we like it or not), FF DH and FF DSA are still very widespread.

- TLS
- SSH
- IPsec
- ...

Various measurements show their endured prevalence.

Who says which are the primes we use?

For a given key size, it should be fine if everybody uses the same p.

It is almost "One prime to rule them all"

De facto: a few primes are very widespread, promoted by:

- Standards (RFCs, …).
- Implementations (Apache, OpenSSL, …), or manufacturers of dedicated equipment (Cisco, Juniper, …).

Who has a say on what primes go there?

The 1992 controversy

Beginning of the 1990s = early days of DSA.
Year 1992: panel at Eurocrypt, CACM article in July, article by Gordon at Crypto.

Is it a good idea to standardize primes?

Most important points raised by (Lenstra and) McCurley:

> So far, it has not been demonstrated that trapdoor moduli for the discrete logarithm problem can be constructed such that a) they are hard to detect, and b) knowledge of the trapdoor provides a quantifiable computational advantage for parameter sizes that could actually be computed by known methods, even with foreseeable machines.
>
> —K. S. McCurley, EC92 panel.

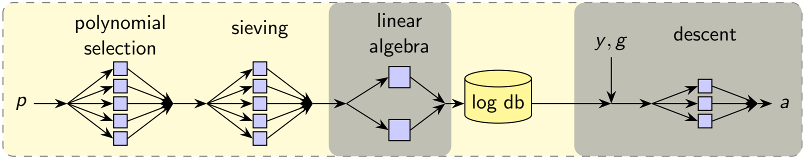
Part of the 1992 discussions focused on why a lower bound on p should be 1024 bits, not 512.

But the above points seemed to suffice to settle the discussion on the trapdoor: too conspicuous, and not a game-changer anyway.

1992 context

In 1992, NFS was still a new algorithm.

- Many practical challenges were yet to be solved.
- Linear algebra appeared a daunting task.
- This is even more true for NFS-DL: first preprint in April 1990.
- Algorithms for individual logs in NFS-DL took years to settle.



All these hurdles have long been passed.

Interlude

Some of the implications of the practice of NFS-DL took a long time to percolate and reach the use of FF-DLP in practice.

Until Logjam, many people overlooked the difference between precomputation (offline) and individual log (online) time for NFS-DL.

		Precomputation core-years	Individual Log core-time
RSA-512	[Cavallar et al. 1999]	1	—
DH-512	[Adrian et al. 2015]	10	10 mins
RSA-768	[Kleinjung et al. 2009]	1,000	—
DH-768	[Kleinjung et al. 2016]	5,000	2 days
RSA-1024	(estimate)	1,000,000	—
DH-1024	(estimate)	$\approx$10,000,000	30 days

What does it look like *now*? (mid-2016)

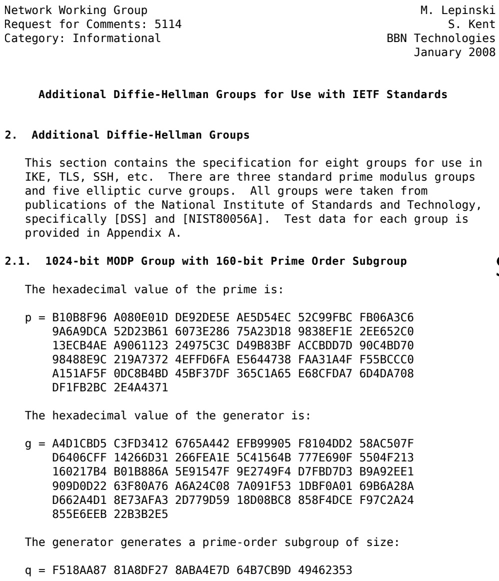
Many primes are found in the wild with unknown provenance.

We cannot tell whether they have been chosen with malice.

- 1024-bit primes in Apache http software;
- RFC 5114 primes ($\geq$1024 bits);
- 2048-bit prime used in IACR 2015 BOD election;
- . . .

We wish to investigate how trapdoors can be designed, and how easier they make the DLP computations.

RFC5114



```
Network Working Group                                       M. Lepinski
Request for Comments: 5114                                      S. Kent
Category: Informational                                BBN Technologies
                                                           January 2008


          Additional Diffie-Hellman Groups for Use with IETF Standards


2.  Additional Diffie-Hellman Groups

   This section contains the specification for eight groups for use in
   IKE, TLS, SSH, etc.  There are three standard prime modulus groups
   and five elliptic curve groups.  All groups were taken from
   publications of the National Institute of Standards and Technology,
   specifically [DSS] and [NIST80056A].  Test data for each group is
   provided in Appendix A.

2.1.  1024-bit MODP Group with 160-bit Prime Order Subgroup

   The hexadecimal value of the prime is:

   p = B10B8F96 A080E01D DE92DE5E AE5D54EC 52C99FBC FB06A3C6
       9A6A9DCA 52D23B61 6073E286 75A23D18 9838EF1E 2EE652C0
       13ECB4AE A9061123 24975C3C D49B83BF ACCBDD7D 90C4BD70
       98488E9C 219A7372 4EFFD6FA E5644738 FAA31A4F F55BCCC0
       A151AF5F 0DC8B4BD 45BF37DF 365C1A65 E68CFDA7 6D4DA708
       DF1FB2BC 2E4A4371

   The hexadecimal value of the generator is:

   g = A4D1CBD5 C3FD3412 6765A442 EFB99905 F8104DD2 58AC507F
       D6406CFF 14266D31 266FEA1E 5C41564B 777E690F 5504F213
       160217B4 B01B886A 5E91547F 9E2749F4 D7FBD7D3 B9A92EE1
       909D0D22 63F80A76 A6A24C08 7A091F53 1DBF0A01 69B6A28A
       D662A4D1 8E73AFA3 2D779D59 18D08BC8 858F4DCE F97C2A24
       855E6EEB 22B3B2E5

   The generator generates a prime-order subgroup of size:

   q = F518AA87 81A8DF27 8ABA4E7D 64B7CB9D 49462353
```

Here is p
Here is $q \mid (p-1)$
Please use for crypto.

Supported by:

- 900K (2.3%) HTTPS hosts
- 340K (13%) IPsec hosts

Plan

Quick NFS recap

To attack FF-DLP, we use NFS-DL.
How do we create a trapdoor that eases NFS-DL?

A quick summary of NFS-DL for a given p:

- Find $f, g \in \mathbb{Z}[x]$ irreducible with $\mathrm{Res}(f, g) = p$.
- Find many $a, b \in \mathbb{Z}$ such that $\mathrm{Res}(f, a - bx)$ and $\mathrm{Res}(g, a - bx)$ are both smooth.
- Solve huge linear system modulo $p - 1$.

Key points

The bitsize of $\mathrm{Res}(\{f, g\}, a - bx)$ and hence the degree and coefficient size of f and g matter enormously.

NFS-DL faster if exceptionally “small” f and g can be found.

NFS goes very well in special cases

For arbitrary p (or N for factoring), there's a lower bound on how small f and g can be (e.g. by counting).

Factoring knows about especially easy integers

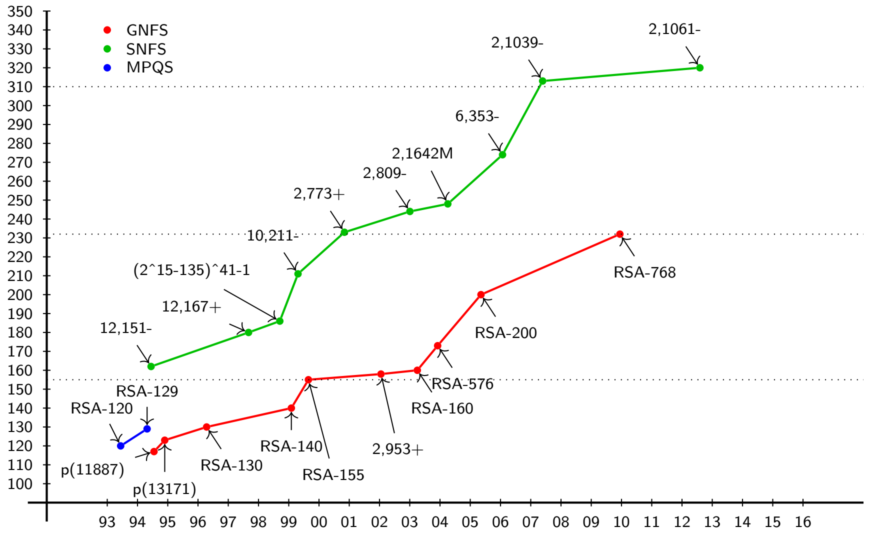
Say if $N = r^e - s$ with r, s small. We pick:

- $f = r^{e \bmod k} X^k - s$ with small k to our liking,
- and $g = X - r^{\lfloor e/k \rfloor}$

This is the special NFS (SNFS, as opposed to GNFS).
Applies in particular to the Cunningham tables.
Likewise, we have an SNFS-DL for "attacker-friendly primes".

Next: timeline of factoring records for SNFS and GNFS, compared.

SNFS versus GNFS (factoring) records



We may ease our task even more

DLP mod attacker-friendly primes may be well within reach while DLP mod “normal” primes of the same size is still remote.

But there is more !

So-called DSA primes

DSS promotes primes with a moderate size subgroup of $(\mathbb{Z}/p\mathbb{Z})^*$
E.g. 1024-bit prime p with 160-bit prime q dividing $p-1$.
RFC5114 promotes examples of such primes.

If a DSA prime is also attacker-friendly, then (S)NFS-DL linear algebra is modulo q, not modulo $p-1$. This is an additional win for the attacker.

Fantasy of a body tinkering with standards

What if we can design attacker-friendly DSA primes?

Heidi hides her polynomials

Heidi, a mischievous protocol designer

- chooses secret polynomials f and g;
- publishes $p = \mathrm{Res}(f, g)$ and pushes for its widespread use.
- p has a (say) 160-bit prime factor q.
- Knowing f and g, Heidi can run SNFS-DL.
  Linear algebra is to be done $\bmod q$.

D. Gordon (Crypto 1992): a way to do just that.
This construction is still efficient today.

How to trapdoor a DSA prime [Gordon92]

Want to construct primes p, q such that $q \mid p-1$ and

$$f(x) = f_6 x^6 + \cdots + f_0, \qquad g(x) = g_1 x - g_0$$

such that $p \mid \mathrm{Res(f,g)}$.

Slow algorithm:

1. Choose random f, g.
2. Check if $p = \mathrm{Res(f,g)}$ prime.
3. Factor $p-1$ with ECM.
4. Repeat until $p-1$ has 160-bit prime factor.

How to trapdoor a DSA prime [Gordon92]

Want to construct primes p, q such that $q \mid p-1$ and

$$f(x) = f_6x^6 + \cdots + f_0, \qquad g(x) = g_1x - g_0$$

such that $p \mid \mathrm{Res(f,g)}$.

Better algorithm:

1. Choose $f(x)$, q, g_0.
2. Want $q \mid \mathrm{Res}(f(x), g_1x - g_0) - 1$.
3. Compute $G(g_1) = \mathrm{Res}(f(x), g_1x - g_0) - 1$.
4. Compute root $G(r) \equiv 0 \bmod q$; $g_1 = r + cq$.
5. Repeat until $\mathrm{Res}(f(x), g_1x - g_0)$ prime.

Plan

$(\mathbb{Z}/p\mathbb{Z})^*$ in crypto

Backdooring primes

Can one unveil the trapdoor?

Computing DL mod 1024-bit primes with Cado-NFS

Outcome and lessons

Can we tell whether p has a trapdoor?

This looks nice for Heidi, but won't work if the primes she pushes for is conspicuously weird.

E.g. you shouldn't do DLP in $(\mathbb{Z}/p\mathbb{Z})^*$ for $p = 2^{1024} - 105$.

However if Heidi allows herself sufficient freedom in choosing the coefficients of f, then p looks innocuous.

Detecting the trapdoor

- "Easy" if $g(x) = x + g_0$ or similar.
  1. Brute force leading coefficient f_d of f.
  2. Search values of g_0 near $(p/f_d)^{1/d}$.
  3. Use LLL to search for other small coefficients of f.

- If $g(x) = g_1 x + g_0$ don't know a way that doesn't require brute forcing coefficients of f or g.

- **Open Problem**: Given $p = \mathrm{Res}(f, g_1 x + g_0)$ and f has small coefficients, find f, g.

Crafting the trapdoor

- 1992-era parameters: 512-bit p, 160-bit q
  - Forces $\deg f = 3$; suboptimal for NFS.
  - f chosen from small set so not well hidden.

> … this trap only makes sense for primes up to [600 bits]. Furthermore, this kind of trap can be detected, although this requires more work than an average user will be able to invest.
>
> —A. Lenstra, EC92 Panel.

- DSA standard: optional "verifiably random" prime generation.

Crafting the trapdoor in the modern era

Gordon's trapdoor construction remains best construction.

- Modern parameters: 1024-bit p, 160-bit q
  - Can choose $\deg f = 6$, optimal for NFS.
  - Choose $|f_i| \approx 2^{11}$.
  - Brute force search to find $f \approx 2^{80} \approx$ cost of Pollard rho for q.
  - Don't know of better way to detect trapdoor.

Exploiting the trapdoor in the modern era

We generated a target 1024-bit prime in 12 core-hours.

The public part:

$$\begin{aligned} p &= 16332398724044367910140207009304915503098943980691751 \\ &\quad 91735800707915692277289328503584988628543993514237336 \\ &\quad 97660534800194492724828721314980248259450358792069235 \\ &\quad 99182658894420044068709413666950634909369176890244055 \\ &\quad 53414932372965552542473794227022215159298376298136008 \\ &\quad 1208200612403808946361023923615765125218049 1 \\ q &= 1120320311183071261988433674300182306029096710473\ , \end{aligned}$$

and Heidi's hidden polynomials:

$$\begin{aligned} f &= 1155\,x^6 + 1090\,x^5 + 440\,x^4 + 531\,x^3 - 348\,x^2 - 223\,x - 1385 \\ g &= 567162312818120432489991568785626986771201829237408\,x \\ &\quad -663612177378148694314176730818181556491705934826717\ . \end{aligned}$$

Plan

NFS-DL with Cado-NFS

We used Cado-NFS to do the DL computations.

- Complete, LGPL-licensed NFS and NFS-DL implementation;
- developed in Nancy since 2007;
- 14,000 commits. 230,000 lines of C and C++ code;
- Used for several DL records.

Common pitfall with NFS computations

Parameters are hard to guess right on first try;
Matrix size often a wild guess.

E.g. for RSA-768, we expected a matrix with 250M to 300M rows, while we got one with 192M rows only.

Predicting computation time

For this computation, we ran tests ahead of time (including a test 768-bit computation).

- Generate sample relations;
  Stir gently to build many fake relations.
- Run the complete filtering suite to build a fake matrix with realistic size.
- Do some test runs for linear algebra on actual hardware, so as to obtain realistic linear algebra timings.

Staged runs as a means to select parameters

NFS parameter selection is somewhat of a dark art.

- Here, size of norms is roughly the same on both sides $\Rightarrow$use special-q's on both sides.
- We want to over-sieve so as to reduce matrix size.
- We put some cofactoring pressure: $2{+}3$ / $3{+}2$ LPs.

Fake relations are ultimately useful in order to:

- adjust sieving parameters;
  - Find appropriate large prime bound;
  - Find appropriate cofactor bound.
- estimate matrix size.
  - adjust various internal filtering parameters.

Preflight predictions

We anticipated (June 24, 2016) 400 core-years total (sieving+LA); and a matrix with 28.5M rows.

Caution: we had never lived to such a promise before.

Preflight predictions

We anticipated (June 24, 2016) 400 core-years total (sieving+LA); and a matrix with 28.5M rows.
Caution: we had never lived to such a promise before.

Sieving started end of June 2016.

- Nancy + UPenn + Grid'5000 (best-effort) ≈3000 cores
- One server per special-$\mathfrak{q}$ side (we had $\mathfrak{q}$'s on both sides).
- Summer also means vacation !
  Jobs ran mostly unattended, and mostly fine.
  (worst, a few SSH tunnels dropped).

Called it a day Aug. 1st, 2016.

```
Final matrix has N=28151570 nc=28151567 (3) w(M)=5630314056
```

Preflight predictions

We anticipated (June 24, 2016) 400 core-years total (sieving+LA); and a matrix with 28.5M rows.
Caution: we had never lived to such a promise before.

Sieving started end of June 2016.

- Nancy + UPenn + Grid'5000 (best-effort) ≈3000 cores
- One server per special-$\mathfrak{q}$ side (we had $\mathfrak{q}$'s on both sides).
- Summer also means vacation !
  Jobs ran mostly unattended, and mostly fine.
  (worst, a few SSH tunnels dropped).

Called it a day Aug. 1st, 2016.

```
Final matrix has N=28151570 nc=28151567 (3) w(M)=5630314056
```

Not so bad.

Part two: linear algebra, block Wiedemann

Wiedemann
- $\{a_i = x^T M^i y \in \mathbb{F}_p\}$ (sequence)
- $\rightsquigarrow$ linear generator F (generator)
- $\rightsquigarrow$ solution $w = F(M)y$ (solution)

$2N + N$ matrix-times-vector products (sequence + solution).

Block Wiedemann: x, y become blocks: $\mathbf{x} \in \mathbb{F}_p^{N\times m}, \mathbf{y} \in \mathbb{F}_p^{N\times n}$.

- $\mathbf{a}_i \in \mathbb{F}_p^{m\times n}$; $n \times (\frac{N}{m} + \frac{N}{n})$ iterations to compute ; n-fold parallel.
- generator $\mathbf{F} = \begin{pmatrix} F_{0,0} & \cdots & F_{0,n-1} \\ \vdots & & \vdots \\ F_{n-1,0} & \cdots & F_{n-1,n-1} \end{pmatrix}$, $\deg F_{i,j} \approx N/n$.
- solution: up to n solutions in $n \times \frac{N}{n}$ iterations, easily parallel.

$\Rightarrow (2 + n/m)N$ matrix-times-vector products, but better distribution opportunities.

Improving on the solution step

Solutions given by columns of **F**: $w_j = \sum_{i=0}^{n-1} F_{i,j}(M) y_i$.

An approach that gives all n solutions:

- Compute the contributions of y_0 to y_{n-1} separately.
- Reuse the $M^k y_i$ that were periodically saved as checkpoints in sequence step $\Rightarrow$ practically unlimited distribution.

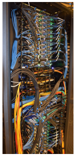
Better approach, for r solutions (Kaltofen95)

- Factor in the $\sum$ on i, and use Horner evaluation.
- Can do it piecewise and reuse the same checkpoints as above. $\Rightarrow$ practically unlimited distribution.
- We need only N/n matrix-times-vector products per solution.

Need $(1 + n/m + r/n)N$ matrix-times-vector products for r solutions.

Computation timings

Linear algebra was done on higher-end hardware with fast interconnect (Infiniband FDR 56Gbps, Cisco UCS 40Gbps)



Used parameters $m = 24$, $n = 12$ for block Wiedemann.

	sieving	linear algebra			individual log
		sequence	generator	solution	
cores	≈3000	2056	576	2056	500–352
CPU time (core)	240 years	123 years	13 years	9 years	10 days
calendar time	1 month	1 month			80 minutes

Computation went smoothly, of course

On the bright side, our computation took almost exactly the predicted time (both CPU time and wall-clock time).

Yet we did have our share of mishaps.

- UPenn: deal with cluster being kicked out of the computer room with 2-day notice, and moved 2 miles south with no decent network connection.
  raspberry pi's + university wifi + ...
- Nancy: of course the improvement of the solution step wasn't coded yet when we started...
  It is now in `cado-nfs-2.3`.

Comparison with other computations

Our computation: $\log_2 p \approx 1024$, $\log_2 q \approx 160$: 400 core-years.

Safe prime of the same size: expect lin.alg $7\times$ harder.

768-bit GNFS-DLP (Kleinjung et al., 2017): $\approx$ 5000 core-years.

2048-bit trapdoored p, like here: expect similar to GNFS-1340.

Some conspicuous SNFS primes found in the wild ($q = (p-1)/2$):

- $p = 2^{1024} - 1093337$: doable but harder than our p!
  - polynomial not as good as ours: α value is bad; sieving $3\times$ harder
  - linear algebra mod $q = (p-1)/2$.
- $p = 2^{784} - 2^{28} + 1027679$ (exercise) $\approx$ 60 core-years.

Plan

$(\mathbb{Z}/p\mathbb{Z})^*$ in crypto

Backdooring primes

Can one unveil the trapdoor?

Computing DL mod 1024-bit primes with Cado-NFS

Outcome and lessons

Danger of over-interpreting the result

We have found no poorly-hidden trapdoored prime in the wild.

- either because the trap was well hidden (after all, the recipe dates back to 1992).
- or because there was no trapdoor at all.

If Heidi designed RFC5114 and suggested the primes used in Apache and so on, she might be caught red-handed in the future. There is no plausible deniability.

Not clear that Heidi is at ease about such a scenario.

NIST encourages IETF to ditch RFC5114

Some talk on the IETF saag mailing list in Oct-Nov. 2016, e.g.

```
From: Tim Polk <wtpolk at gmail.com>
To: saag@ietf.org
Date: Fri, 4 Nov 2016 11:11:26 -0400
Subject: Provenance of Diffie-Hellman groups in RFC 5114

Folks,

The three Diffie-Hellman groups included in RFC 5114 were originally used by NIST to create
test vectors to validate implementations, nothing more, and certainly not as a recommendation
for people to use or adopt them operationally.

We were not at that time concerned about trap doors in test vectors since we did not expect
operational use of these groups.  For operational use, traceability of generation is an
important best practice.  After some searching through our records and old source files, NIST
cannot determine specifically how these Diffie-Hellman domain parameters were generated,
although we think that they were generated internally at NIST.

NIST sees no need to standardize or recommend these specific Diffie-Hellman groups for any use
other than testing.  We believe it is important that the provenance of any critical domain
parameters recommended or required by a standard be fully explained.  Therefore it would be
appropriate for the IETF to remove or deprecate any inclusion of these groups in an RFC.

Thanks,

Tim Polk
```

RFC8247 (09/2017): this just happened

Algorithm Implementation Requirements and Usage Guidance for [IKEv2].

Number	Description	Status
14	2048-bit MODP Group	MUST
19	256-bit random ECP group	SHOULD
5	1536-bit MODP Group	SHOULD NOT
2	1024-bit MODP Group	SHOULD NOT
1	768-bit MODP Group	MUST NOT
22	1024-bit MODP Group with 160-bit Prime Order Subgroup	MUST NOT
23	2048-bit MODP Group with 224-bit Prime Order Subgroup	SHOULD NOT
24	2048-bit MODP Group with 256-bit Prime Order Subgroup	SHOULD NOT

```
Groups 22, 23, and 24 are MODP groups with Prime Order Subgroups that are not safe
primes.  The seeds for these groups have not been publicly released, resulting in
reduced trust in these groups.  These groups were proposed as alternatives for
groups 2 and 14 but never saw wide deployment.  It has been shown that group 22
with 1024-bit MODP is too weak and academia have the resources to generate
malicious values at this size.  This has resulted in group 22 to be demoted to
MUST NOT.  Groups 23 and 24 have been demoted to SHOULD NOT and are expected to
be further downgraded in the near future to MUST NOT.  [...]
```

Lessons

1024-bit DLP can be easy for an attacker that maliciously chose the prime to his liking.

We found no easy way to prove that a trapdoor is present.

Verifiable randomness is necessary.

- It's not even the question of accusing anyone of wrongdoing. We found no smoking gun.
- But the lack of verifiable randomness is a major hindrance for trust in cryptographic standards.

Of course **people still get it awfully wrong**.

E.g. the standardized French and Chinese elliptic curves are really really bad to this regard.

More details

A kilobit hidden SNFS discrete logarithm computation.
Joshua Fried, Pierrick Gaudry, Nadia Heninger, and Emmanuel Thomé. https://eprint.iacr.org/2016/961 (Eurocrypt 2017).