A kilobit hidden SNFS discrete log computation

Joshua Fried, Pierrick Gaudry, Nadia Heninger,
Emmanuel Thomé

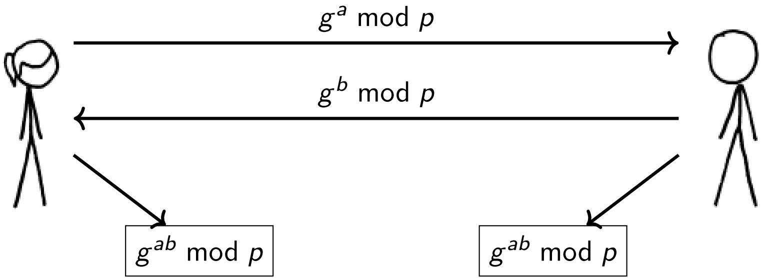
May 1, 2017

Textbook (Finite-Field) Diffie-Hellman Key Exchange

[Diffie Hellman 1976]

p a prime (so $\mathbb{F}_p^*$ is a cyclic group)

$g < p$ group generator (often 2 or 5)



Images from XKCD

Where do group parameters come from?

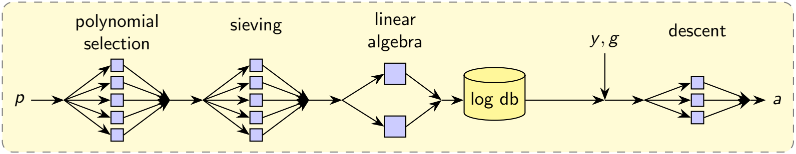
- Protocol Specifications (RFCs)
  - TLS 1.3, SSH, IPsec (IKE)
- Distributed in implementations
  - Apache webserver, OpenSSH server, Java JDK
- Generated by users
  - Possible in SSH and TLS prior to version 1.3
  - 80% of TLS hosts use 1 of 10 primes

Our work

1. What does backdooring a prime look like?
2. Is it detectable?
3. What sort of computation would be required today?
4. Impact for currently deployed crypto

Number field sieve discrete log algorithm

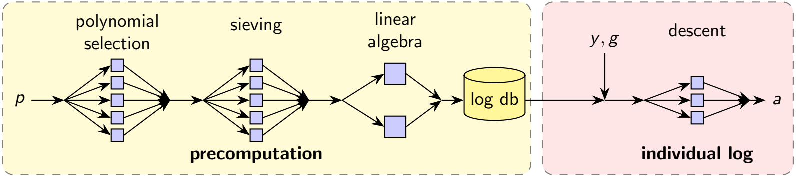
[Gordon], [Joux, Lercier], [Semaev]



1. **Polynomial selection**: Find a good choice of number field K.
2. **Relation collection**: Factor elements over $\mathcal{O}_K$ and over $\mathbb{Z}$.
3. **Linear algebra**: Once there are enough relations, solve for logs of small elements.
4. **Individual log**: "Descent" Try to write target t as sum of logs in known database.

How long does it take to compute discrete logs?

(For the “general” number field sieve)

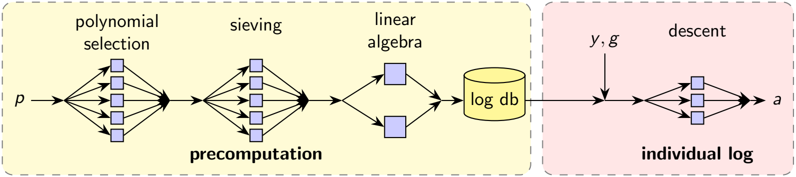


Answer 1:

$$L_p(1/3, 1.923) = \exp(1.923(\log p)^{1/3}(\log\log p)^{2/3})$$

How long does it take to compute discrete logs?

(For the "general" number field sieve)

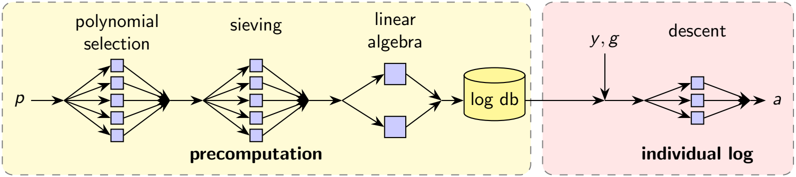


Answer 1:

$$L_p(1/3, 1.923) = \exp(1.923(\log p)^{1/3}(\log\log p)^{2/3}) \qquad L_p(1/3, 1.232)$$

How long does it take to compute discrete logs?

(For the "general" number field sieve)



Answer 2:

		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)	≈10,000,000	30 days

Polynomial selection for the number field sieve

"Easy" Polynomial Selection

1. Choose $m \approx p^{1/6}$. Write p in base m:

$$p = f_6 m^6 + f_5 m^5 + \cdots + f_0$$

2. Then a suitable pair of polynomials for NFS is

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

   f, g share common root $\bmod p$.

3. Expect $|f_i| \approx |p^{1/6}|$.

4. Size of numbers to be sieved depends on $|f_i|, m$. Smaller size $\rightarrow$ higher probability of being B-smooth $\rightarrow$ less work to find each relation.

The "special" number field sieve

Even easier polynomial selection!

1. Consider Mersenne number $n = 2^k - 1$.
2. Assume $6 \mid k$. Let $m = 2^{k/6}$ so we have $f(x) = x^6 - 1$ and $g(x) = x - m$.

Impact for discrete log:

	GNFS core-years	SNFS core-years
Asymptotically	$L_p(1/3, 1.923)$	$L_p(1/3, 1.526)$
DH-768	5,000	60
DH-1024	≈10,000,000	400

Flashback to the crypto wars of the 1990s

- 1991: NIST proposed draft standard for discrete log-based Digital Signature Algorithm (DSA)

  Params:

  - p 512-bit prime modulus
  - g generates subgroup of 160-bit prime order q

- A. Lenstra: Primes can be trapdoored if they include hidden SNFS structure.

How to trapdoor a DSA prime.

[Gordon 92]

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

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.

[Gordon 92]

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

Better algorithm:

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

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.

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, Eurocrypt 1992 Panel on DSA

- 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

1. Generated target prime in 12 core-hours.

$$
\begin{aligned}
p &= 1633239872404436791014020700930491550309894398069175 1\\
&\ 9173580070791569227728932850358498862854399351423733 6\\
&\ 9766053480019449272482872131498024825945035879206923 5\\
&\ 9918265889442004406870941366695063490936917689024405 5\\
&\ 5341493237296555254247379422702221515929837629813600 8\\
&\ 120820061240380894636102392361576512521804 91\\
q &= 1120320311183071261988433674300182306029096710473\ ,
\end{aligned}
$$

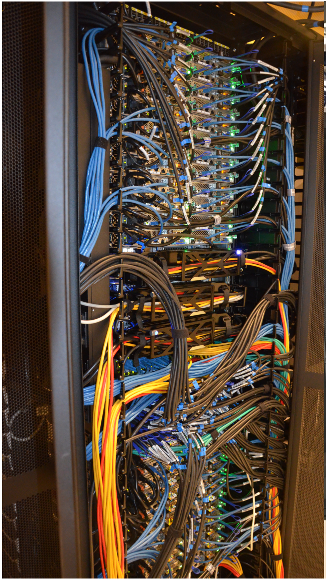
$$
\begin{aligned}
f &= 1155\,x^6 + 1090\,x^5 + 440\,x^4 + 531\,x^3 - 348\,x^2 - 223\,x - 1385\\
g &= 567162312818120432489991568785626986771201829237408\,x\\
&\ -663612177378148694314176730818181556491705934826717\ .
\end{aligned}
$$

Exploiting the trapdoor in the modern era

2. Run discrete log computation mod p.

	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

INRIA Catrel



UPenn

Exploiting the trapdoor in the modern era

3. Are there SNFS primes in the wild?

Exploiting the trapdoor in the modern era

3. Are there SNFS primes in the wild?

Non-hidden: yes.

Prime	NFS time # cores	Source
$p = 2^{512} - 38117$	215 minutes 1288 cores	Internet Scanning 121 TLS hosts
$p = 2^{784} - 2^{28} + 1027679$	23 days 1000 cores	LibTomCrypt
$p = 2^{1024} - 1093337$	$\approx$ 6 months 2000 cores	Internet Scanning 125 TLS hosts

Exploiting the trapdoor in the modern era

3. Are there SNFS primes in the wild?

Poorly-hidden: no.

- We did a somewhat perfunctory search for primes with $g_1 = 1$ and 10-digit f_i. Did not find any.

Provenance of Diffie-Hellman groups in the wild

- Verifiably Random
  - Java JDK primes have published seeds
- "Nothing up my sleeve"
  - Oakley groups - generated from digits of π
  - TLS 1.3 groups - generated from digits of e

Provenance of Diffie-Hellman groups in the wild

- Verifiably Random
  - Java JDK primes have published seeds
- "Nothing up my sleeve"
  - Oakley groups - generated from digits of π
  - TLS 1.3 groups - generated from digits of e
- No record of provenance
  - Groups published in RFC 5114
  - Groups included with Apache webserver

[Docs] [txt|pdf] [draft-lepinski-dh...] [Diff1] [Diff2]

INFORMATIONAL

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
```

Supported by:

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

Provenance of Diffie-Hellman groups in RFC 5114

"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.

. . . it would be appropriate for the IETF to remove or deprecate any inclusion of these groups in an RFC." — Tim Polk, November 2016

What about 2048 bits?

Gordon's trapdoor construction would work.

- Modern parameters: 2048-bit p, 224 or 256-bit q
  - Can choose $\deg f = 7$, optimal for NFS.
- Estimate 2048-bit SNFS is roughly equivalent to 1340-bit GNFS
  - ($\approx$ 7,000,000,000 core years)

Design considerations for future algorithms

- Eliminate potential for backdoored parameters.
  - Even if Dual-EC was never backdoored by the NSA, someone exploited the potential backdoor against Juniper.
- If verifiable randomness is necessary, it should not be considered optional.
- Account for precomputation in analysis.

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