Cryptography

5 – Public-key encryption II: Discrete logarithms

G. Chênevert

October 21, 2019

ISEN
ALL IS DIGITAL!
LILLE

yncréa

Today

Modular DLP

Applications

Cryptanalysis

Modular exponentiation

For a given modulus n:

$$(g, \xi) \;\mapsto\; x \underset{n}{\equiv} g^{\xi}$$

RSA: hard to recover g from x even if ξ is known

(essentially need to factor n)

"discrete ξ^{th} root problem"

Discrete logarithm problem

Also hard to recover ξ from x even if g is known!

Definition (Discrete logarithm)

$$\log_g x :\underset{\nu}{\equiv} \xi \iff x \underset{n}{\equiv} g^{\xi},$$

where ν is the **multiplicative order** of g, *i.e.* the smallest positive integer for which

$$g^{\nu} \underset{n}{\equiv} 1.$$

By Fermat's theorem, we know in general that $\nu \mid \varphi(n)$.

General fact

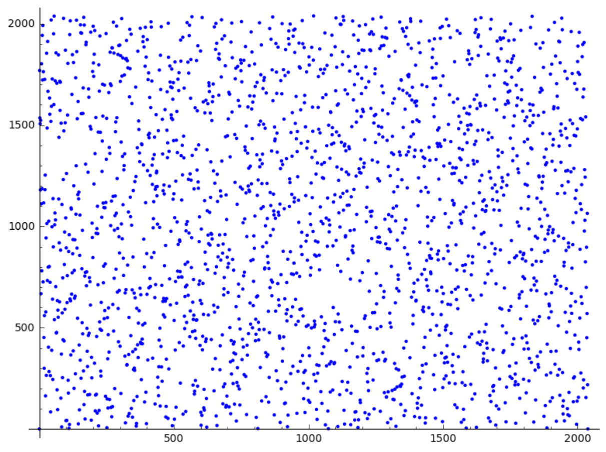
Logarithms never behave quite as well as exponentials.

(think: speed of convergence of power series, ...)

Meaning here: discrete logs can take much *longer* to compute than modular exponentials.

Information can be hidden in exponents!

Example: $x \underset{2039}{\equiv} 1769^{\xi}$



Today

Modular DLP

Applications

Cryptanalysis

Secret sharing

Public-key encryption provides a partial solution to the problem of setting up a shared private key for symmetric encryption on an insecure channel:

- Alice chooses secret k,
- encrypts it with Bob's public encryption key,
- and sends it to him;
- Bob recovers k using his private decryption key.

Are there problems with that? (hint: yes)

"Symmetric" version

- Alice chooses k_A and sends it to Bob using his public encryption key;
- Bob chooses k_B and sends it to Alice using her public encryption key;
- Shared secret is $k := k_A \oplus k_B$.

Better: neither Alice nor Bob fully controls the final secret.

But two public encryption key pairs are needed...

Diffie-Hellman (1976)

- Alice and Bob agree on "safe" parameters n and g.
- Alice chooses α, computes $a \underset{n}{\equiv} g^{\alpha}$ and sends it to Bob.
- Bob chooses β, computes $b \underset{n}{\equiv} g^{\beta}$ and sends it to Alice.

Shared secret is

$$k :\underset{n}{\equiv} g^{\alpha\beta} \equiv a^{\beta} \equiv b^{\alpha}.$$

Diffie-Hellman problem

Eve is faced with the problem:

$$\text{given } a \text{ and } b\text{, recover } k.$$

We *believe* that her best line of attack is:

- compute $\alpha = \log_g a$ or $\beta = \log_g b$
- then easily deduce $k \underset{n}{\equiv} g^{\alpha\beta}$.

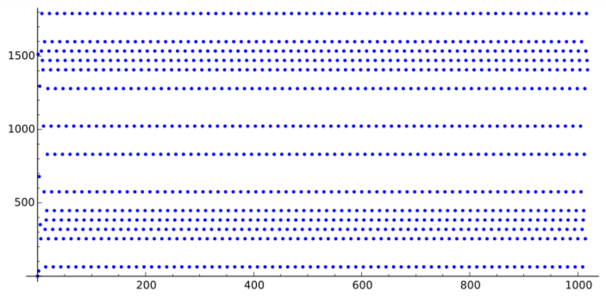
Caveats

- Should **always** be used in conjunction with authentication to prevent *man-in-the-middle attacks*

- Bob should check that Alice does not provide a value of a for which the discrete log is easy

  (same on Alice's side)

Example: $x \underset{1856}{\equiv} 1514^{\xi}$



ElGamal cipher (1985)

Essentially Diffie-Hellman $+$ one-time multiplicative pad

Public parameters: n and g (can be reused)

Keys:

- δ private decryption key
- $e \underset{n}{\equiv} g^{\delta}$ public encryption key

Alice wants to send a message $m \in [\![0, n[\![$ to Bob.

Encryption

- Alice chooses random σ, computes $s \underset{n}{\equiv} g^{\sigma}$
- Computes shared secret $k \underset{n}{\equiv} e^{\sigma}$
- Computes encrypted $c \underset{n}{\equiv} k \cdot m$
- Sends the pair (s, c)

Decryption

Upon reception of a pair (s, c), Bob

- Computes shared secret $k \underset{n}{\equiv} s^{\delta}$
- Recovers $m \underset{n}{\equiv} k^{-1} \cdot c$

Same caveats apply!

Today

Modular DLP

Applications

Cryptanalysis

Attacks on the DLP

or: *how to compute discrete logarithms*

To understand how to choose "safe" parameters n and g we need to understand how to force the DLP algorithms to be in the worst-case scenario.

Naive algorithm: brute-force the exponent

Takes at most $\mathcal{O}(\nu) \leq \mathcal{O}(n)$ steps

$\implies$ want g of large multiplicative order ν (hence large n)

Chinese remainder theorem

If $n = n_1 \cdot n_2$ with n_1 and n_2 coprime:

$$x \underset{n}{\equiv} g^{\xi} \iff \begin{cases} x \underset{n_1}{\equiv} g^{\xi} \\ x \underset{n_2}{\equiv} g^{\xi} \end{cases}$$

If ξ is recovered modulo ν_1 and ν_2, it is then easily recovered modulo $\nu = \mathrm{LCM}(\nu_1, \nu_2)$

$\implies$ n should be as prime as possible

Here, this means: n should be prime

CRT (again)

Hence take n a prime, so that $\varphi(n) = n - 1$.

Remember we are looking for a value $\xi \bmod \nu \mid \varphi(n)$.

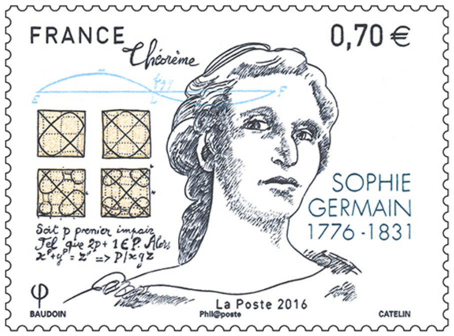
If $\varphi(n) = n - 1$ factors, we can speed up the process by working modulo the factors.

$\implies n - 1$ should be as prime as possible

Here, the best we can do is: $n = 2q + 1$ with q prime

(n: **safe** prime, q: associated **Sophie Germain** prime)

Sophie Germain (1776-1831)



Primitive roots

Fact: For n prime, there exists in $(\mathbb{Z}/n\mathbb{Z})^\times$ an element of order $n-1$.

Hence, for a safe prime:

$$(\mathbb{Z}/n\mathbb{Z})^\times \simeq \mathbb{Z}/(n-1)\mathbb{Z} \overset{\text{CRT}}{\simeq} \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$$

Most nonzero elements g have multiplicative order q or $2q$.

Only two of them generate small subgroups:

$$1 \simeq (0,0) \quad \text{and} \quad -1 \simeq (1,0).$$

Baby-step giant-step

Time/memory trade-off on the naive algorithm to compute $\xi \underset{\nu}{\equiv} \log_g x$.

Pick some base β and write $\xi = i\beta + j$.

Baby step:

Compute and store all powers $g^j \bmod n$ for $j \in [\![0, \beta[\![$ in a table

Giant step:

For every $i \in [\![0, \frac{\nu}{\beta}[\![$, check if $x \cdot (g^{-\beta})^i \bmod n$ is in the above table

Baby-step giant-step

Time complexity: $\mathcal{O}(\beta) + \mathcal{O}(\frac{\nu}{\beta})$

Space complexity: $\mathcal{O}(\beta)$

Often take $\beta \approx \sqrt{\nu}$ to get time and space complexities

$$\mathcal{O}(\sqrt{\nu}).$$

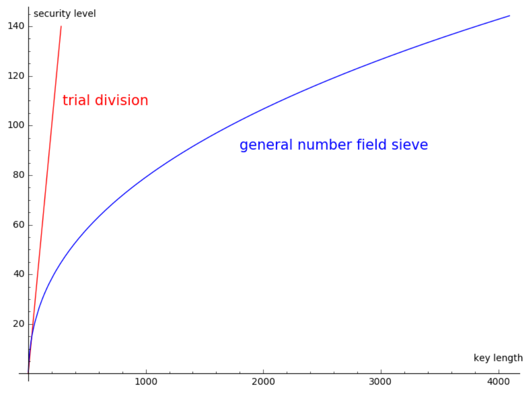
Other algorithms

There also exists a general-purpose probabilistic algorithm that takes (on average) $\mathcal{O}(\sqrt{\nu})$ steps (and $\mathcal{O}(1)$ memory)

The General Number Field Sieve solves the *modular* DLP

$\implies$ use same key lengths as for RSA

Recall



security level
140
120
100
80
60
40
20
trial division
general number field sieve
key length
1000
2000
3000
4000

Generalized DLP

The nice thing about the DLP is that it can be asked in any abelian group $\mathcal{G}$:

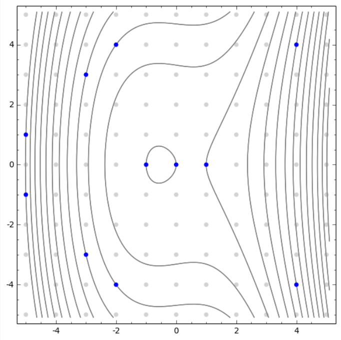
Given $g \in \mathcal{G}$ and x such that

$$x = g^{\xi} = \underbrace{g \cdot g \cdots g}_{\xi} \quad \text{in} \quad \mathcal{G},$$

find $\xi \underset{\nu}{\equiv} \log_g(x)$, with $\nu = \mathrm{ord}_{\mathcal{G}}(g)$.

So far we used $\mathcal{G} = (\mathbb{Z}/n\mathbb{Z})^{\times}$, but there are other interesting groups...

Elliptic curves



Best known DLP algorithms are the *generic* ones

$\Longrightarrow$ ℓ-bit security achieved by 2ℓ-bit keys ☺