T-79.159 Cryptography and Data Security

Lecture 6: Cryptanalysis of public-key algorithms.

Markku-Juhani O. Saarinen
Helsinki University of Technology
mjos@tcs.hut.fi

Outline

- Computational complexity
- Reminder about basic number theory
- Factoring
- Discrete logarithm
- Elliptic curve discrete logarithm

  (Some items are partially repeat of the last lecture)

Computational complexity

I will give asymptotic estimates for the complexity of many of the algorithms. Remember: Most complexity numbers in cryptology are actually *conjectures* (i.e. educated guesses).

The complexity measurements are often based on experimentation and heuristics, and simply give an estimate of the expected running time. Most of these algorithms are not *deterministic*; they can be expected to work in most cases but there may be *pathological* cases where they in fact do not terminate.

However, if you're trying to break a cipher, it doesn't matter what you use as long as it **WORKS**!

Basic complexities

- Modular multiplication: Schoolboy method, roughly $O(l^2)$ (although faster methods exist, based on Karatsuba's method, FFT, etc).
- Greatest common divisor (gcd): Euclid's method, roughly $O(l^3)$.
- Modular inversion: Extended Euclid's method, roughly $O(l^3)$.
- Modular exponentiation: Square-and-multiply, roughly $O(l^3)$.
- Elliptic curve addition: with clever programming, $O(l^2)$.

Elementary number theory

The fundamental theorem of arithmetic (Euclid) states that any positive integer can be represented in exactly one way as a product of prime numbers (if we ignore the order).

e.g. $6 = 2 * 3$, $48 = 2^4 * 3$.

The first primes are $2, 3, 5, 7, 11, 13, 19, \cdots$

Density of primes around n is roughly $1/\ln(n)$ ("Prime number theorem"). This means that if we are looking for, say, 1024-bit prime, there is a probability of $1/\ln(2^{1024}) = 1/709.8$ that a random number is a prime. Hence we must test about 500 candidates before we can truly find a prime (of course we can sieve out even numbers etc, which will speed up the search).

Fermat's "little" theorem

Let p be a prime which does not divide the integer a, then

$$a^{p-1} \equiv 1 \pmod{p}.$$

The inverse is *almost true*; there is an extremely high probability (for large n) that if $2^{n-1} \equiv 1 \pmod{n}$, then n is a prime.

For cryptology probable primes are perfectly acceptable, and e.g. only this Fermat test is used in search of primes.

Complexity of *deterministic primality test* has been recently proved to be Polynomial (Agrawal, Kayal, Saxena: "PRIMES is in P", 2002). However, this $O(l^{7.5})$ algorithm has little practical effect to cryptography.

A Group

A finite group is a set of elements together with a *binary operation* (group operation) that satisfies:

- Closure: $a + b$ is in the set if a and b are in the set.
- Associativity: $(a + b) + c = a + (b + c)$.
- Identity: There exists an element o that satisfies $a + o = o + a = a$.
- Inverse: Every element has an inverse $a - a = o$.

Examples of a group

- Elements 0 and 1 together with XOR operation.
- Addition modulo n, where n is any positive integer.
- Multiplication modulo p, where p is a prime number.
- Group formed by an elliptic curve together with the addition rule (previous lecture).

Factoring

The task of splitting a number n into its prime components $n = pq$.

> "The problem of distinguishing prime numbers from composites and of resolving composite numbers into their prime factors, is one of the most important and useful in all of arithmetic.
>
> . . . The dignity of science seems to demand that every aid to the solution of such an elegant and celebrated problem be zealously cultivated."
>
> – C.F. Gauss, Disquisitiones Arithmeticae, Article 329 (1801)

Factoring and RSA

It is clear that if the public modulus n can be factored, then RSA is broken.

However it has not been proved that if a method exists to solve the RSA problem, it can also be used to factor numbers.

Hence the RSA problem is not equivalent to factoring!

.. but it seems that factoring algorithms are the easiest method of breaking RSA (other than implementation attacks).

Trial division

Even the greatest mathematicians of 1800's still used trial division to factor numbers.

> "Mr. Landry, at the age of 82, after several months work in 1880 obtained the following result:
>
> $2^{64} + 1 = 274177 * 67280421310721$".
>
> – Édouard Lucas, Récrétaions Mathématiques (1891)

He may have used some tools to speed up his hand computation but the complexity of his method was still $O(\sqrt{n})$.

Pollard's ρ factoring

1. $x \leftarrow$ random number $2 \cdots n-1$, $y \leftarrow x$, $k \leftarrow 1$
2. for $i = 1, 2, 3, \cdots$ do:
3. $\quad x \leftarrow (x^2 - 1) \mod n$
4. $\quad d \leftarrow \gcd(y - x, n)$.
5. $\quad$ if $d \neq 1$ and $d \neq n$ print d
6. $\quad$ if $i = k$ then $y \leftarrow x$, $k \leftarrow 2k$.

Pollard's ρ factoring complexity

The choice of polynomial $x^2 - 1$ is arbitrary (but e.g. $x - 1$ or x^2 wouldn't work).

This is an Heuristic method based on the birthday paradox.

If a "match" occurs $\mathrm{mod}\ mod\ p$, then it is caught by the gcd.

Complexity $O(n^{1/4})$ – non-deterministic.

The best known *deterministic* factoring algorithms have the same complexity as this extremely simple algorithm!

First sub-exponential algorithms

Most modern factoring algorithms attempt to find integers x and y such that

$$x^2 \equiv y^2 \pmod{n}$$

In case there's a 50 % chance that $gcd(n, x - y)$ will yield a factor of n.

The algorithms generally consist of two stages.

1. Sieving step: Find "relations" (easily distributable)

2. Matrix step: Linear algebra phase (not easily distributable)

Factoring algorithms

The development has followed the following path:

- CFRAC: Continued fraction algorithm (mid 70s)
- MPQS: Multiple Polynomial Quadratic Sieve (mid 80s)
- GNFS: General Number Field Sieve (mid 90s)

GNFS is the current champion. There is no guarantee that it is the *best possible algorithm*!

GNFS complexity

Current heuristic estimates place the complexity of GNFS at:

$$O(e^{(1.9229+O(1))*\ln(n)^{1/3}*\ln(\ln(n))^{2/3}})$$

Where n is the number to be factored and we assume that it is a product of two large primes ("RSA number").

Note1: It is much easier to factor a *random number* than it is to factor a RSA number. With non-neglible probability it will be a prime and no factorization is necessary!

Note2: GNFS starts to beat MPQS only after $n > 2^{500}$!

Elliptic Curve Method (ECM)

A different approach is taken by the ECM algorithm, which is based on elliptic curves. Its complexity is:

$$O(e^{\sqrt{\ln(p)\ln(\ln(p))}(1+O(1))}$$

Where $p \mid n$ is the smallest factor. In other words, the complexity of ECM is related to the size of the *component*, not the size of the whole *composite* n.

A good policy for factoring large random numbers is to start with ECM and then move to MPQS when we expect the remaining unfactored part to consist of large primes...

Discrete logarithm (DL)

The task of finding the *index* of an element in a finite group.

Example: in the multiplicative group of numbers modulo p, given a and the *generator* g, find the index x that satisfies

$$g^x \equiv a \pmod{p}$$

If g is indeed the generator of the group, such x will always exist and is unique.

Numerous public key algorithms are based on discrete log problem: Diffie-Hellman, ElGamal, DSA, ECDSA, etc..

DL Example

$g = 3$ is the smallest generator when $p = 7$.

$$
\begin{aligned}
3^0 &= 1 \bmod 7 \\
3^1 &= 3 \bmod 7 \\
3^2 &= 2 \bmod 7 \\
3^3 &= 6 \bmod 7 \\
3^4 &= 4 \bmod 7 \\
3^5 &= 5 \bmod 7 \\
3^6 &= 1 \bmod 7 \\
&\ldots
\end{aligned}
$$

We see that the discrete logarithm of 6 with generator 3 is 3.

DL properties

Discrete logs satisfy all the normal properties of logarithms:

$$g^a * g^b \equiv g^{a+b} \quad (1)$$

$$g^a / g^b \equiv g^{a-b} \quad (2)$$

$$(g^a)^b = (g^b)^a \equiv g^{ab} \quad (3)$$

$$g^{-a} * g^a \equiv 1 \quad (4)$$

Observation 3 is in fact at the heart of Diffie-Hellman key exchange..

DL for groups

The obvious trial division algorithm for a general group has $O(n)$ complexity and is clearly not acceptable.

However, a version exists of Pollard's ρ method which can be used to compute discrete logs in $O(\sqrt{n})$ time. This is relatively straight-forward to derive. The method is (again) based on the birthday paradox.

In fact Shoup proved in 1997 that no algorithm can be faster than this, if it is not allowed to utilize properties of the *representation of the group*.

Pollard's ρ DL

The group G is partitioned into three sets S_1, S_2, S_3 of roughly equal size. We define a sequence $x_1, x_2, \cdots$ as:

$$x_{i+1} = \begin{cases} \beta * x_i & \text{when } x_i \in S_1 \\ x_i^2 & \text{when } x_i \in S_2 \\ \alpha * x_i & \text{when } x_i \in S_3 \end{cases}$$

Here α is the generator and β is the number of which discrete logarithm we are trying to determine. This sequence then defines two sequences of integers a_i and b_i satisfying $x_i = \alpha^{a_i}\beta^{b_i}$.

Pollard's ρ DL (2)

$$a_{i+1} = \begin{cases} a_i \mod n & \text{when } a_i \in S_1 \\ 2a_i \mod n & \text{when } a_i \in S_2 \\ a_i + 1 \mod n & \text{when } a_i \in S_3 \end{cases}$$

and

$$b_{i+1} = \begin{cases} b_i + 1 \mod n & \text{when } b_i \in S_1 \\ 2b_i \mod n & \text{when } b_i \in S_2 \\ b_i \mod n & \text{when } b_i \in S_3 \end{cases}$$

Here n is the group size and $a_0 = b_0 = 0$.

Pollard's ρ DL (3)

We use Floyd's cycle-finding algorithm (lecture 4) to find two group elements x_i and x_{2i} such that $x_i = x_{2i}$.

Hence $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}}$, and it follows that $\beta^{b_i - b_{2i}} = \alpha^{a_{2i} - a_i}$. Taking logs we get:

$$(b_i - b_{2i}) * \log_\alpha \beta \equiv (a_{2i} - a_i) \pmod{n}$$

This equation the quickly yields the discrete logarithm $\log_\alpha \beta$.

More on Discrete Logs

The problem of finding discrete logarithms in an elliptic curve group (ECDL) appears to be as hard as DL in general group; $O(\sqrt{n})$ algorithms are the best known ones. However, this has not been proved.

For the problem of finding DL in the group of integers modulo p more efficient methods exist, and they are based on the representation of the group. The methods (Index Calculus method, Number Field Sieve DL) are analogous to the best known factorization methods and their complexity roughly the same.

Some algorithms (most notably DSA) are based on subgroup discrete logarithm method, where it is possible to use the general all-purpose DL algorithms in the subgroup, but also utilize the representation. DSA is designed to make the security margin against both of these attacks similar (1024-bit p, 160-bit subgroup).

Putting it all together

Symmetric	Hashes	RSA	Elliptic curve
56	112	417	112
64	128	682	128
80	160	1464	160
100	200	3137	200

Key sizes in each row correspond to each other in terms of security.

Currently a 2048-bit RSA key, 128-bit AES key, 256-bit SHA256 hash and 256-bit elliptic curve fields are considered secure for all foreseeable future.

Note: The security of the cryptosystem (as a whole) = the security of the weakest component.