

Lecture 8

Computational number theory

- Prime numbers
- Factoring methods
- Chinese Remainder Theorem
- Algorithms for discrete logarithms

CHALMERS

More number theory – why?

The topics for today have cryptographic significance in several ways:

- Cryptanalysis of public key primitives is finding new algorithms in number theory (for factoring, discrete log ...).
- Complexity of algorithms for factoring and discrete logs determine recommended key sizes. An idea of these algorithms helps in understanding recommendations.
- Prime number tests are important ingredients in cryptographic libraries. One should have an idea of how they work.
- CRT is useful in several ways in cryptography, e.g. in optimizing RSA.

CHALMERS

Prime numbers

Will we run out of primes?

Cryptographic applications need a large supply of prime numbers.

Let $\pi(n)$ = the number of primes $\leq n$.

Theorem (The prime number theorem)

For large n

$$\pi(n) \approx \frac{n}{\ln n}$$

Thus, the probability that a randomly chosen 1024-bit number is prime is approximately 1/700.

So, a prime can be generated by picking random numbers until one passes a test for being prime.

How does one do that test?

Prime numbers

Probabilistic tests

The problem to decide whether a given integer N is a prime or not was recently (2002) shown to be in **P** by the Indian computer scientist Manindra Agrawal and two of his students.

However, this has not yet resulted in efficient tests.

In practice, one uses **probabilistic** tests.

These may **erroneously** claim that a composite number is prime, but the probability for this can be made arbitrarily small.

CHALMERS

Fermat's test

Recall that, if p is prime and $a \in \mathbb{Z}_p^*$, then $a^{p-1} \bmod p = 1$.

Fermat's test for primeness of n :

- ① Pick random a and compute $a^{n-1} \bmod n$.
- ② If result is $\neq 1$, declare n composite.
- ③ Repeat the above steps t times; if all results are 1, declare n prime.

By choosing t suitably, one hopes to bound probability of erroneously declaring n prime.

CHALMERS

Carmichael numbers

But, unfortunately, there are (rare) composite numbers n , called **Carmichael numbers**, for which $a^{n-1} \equiv 1 \pmod{n}$ for all $a \in \mathbb{Z}_n^*$.

The three smallest Carmichael numbers are 561, 1105, 1729.

There are infinitely many Carmichael numbers.

Carmichael numbers are rare, but not rare enough to be ignored, so we need to improve Fermat's test.

Example

We consider $n = 561$ and note that $560 = 35 \cdot 2^4$. Thus $a^{560} = (((a^{35})^2)^2)^2$. Better, we put

$$\begin{aligned}x_0 &= a^{35} \bmod n \\x_{k+1} &= x_k^2 \bmod n, \quad k = 0, 1, 2, \dots\end{aligned}$$

and note that $a^{560} \bmod n = x_4$.

Two examples

Example

Intermediate results, when computing $a^{560} \bmod 561$ for some a :

a	x_0	x_1	x_2	x_3	x_4
2	263	166	67	1	1
3	78	474	276	441	375
4	166	67	1	1	1
5	23	529	463	67	1
11	209	484	319	220	154
13	208	67	1	1	1
35	494	1	1	1	1
52	307	1	1	1	1

Example

Now, let us consider $n = 569$ (which is prime). Note that $568 = 71 \cdot 2^3$ and put $x_0 = a^{71}$.

a	x_0	x_1	x_2	x_3
2	86	568	1	1
3	277	483	568	1
4	568	1	1	1
5	1	1	1	1
11	493	86	568	1
13	86	568	1	1
35	86	568	1	1
52	483	568	1	1

Can you see any patterns?

Square-roots of 1

Fact 1

Assume that p is a prime and that $x^2 = 1 \bmod p$. Then either $x = 1 \bmod p$ or $x = p - 1 \bmod p$.

Proof.

We have that $x^2 - 1 = 0 \bmod p$, hence

$$(x+1)(x-1) = kp$$

(in \mathbb{Z}) for some k .

Since p is a prime, it must be a factor in either $x - 1$ or $x + 1$. In the former case, $x = 1 \bmod p$, in the latter $x = p - 1 \bmod p$. \square

Note

$p - 1 = -1 \in \mathbb{Z}_p^*$, so the square-roots of 1 are 1 and -1 .

Further facts

Fact 2 (not proved here; not difficult)

If n is odd and has r different prime factors, then there are 2^r numbers $x \in \mathbb{Z}_n^*$ such that $x^2 = 1 \in \mathbb{Z}_n^*$.

Fact 3 (not proved here; quite difficult)

Let n be odd **and composite** and $n - 1 = 2^s \cdot m$ where m is odd.

For given a , we compute a^{n-1} as x_s , where $x_0 = a^m$ and $x_{k+1} = x_k^2$ for $k = 0, 1, \dots$

For at least $3/4$ of the possible a ($1 \leq a \leq n - 1$), the sequence $x_0, x_1 \dots x_{s-1}$ will include a square-root of 1 **different from** ± 1 .

Algorithm idea

Try t randomly chosen values for a ; if you never encounter a squareroot of 1 that is $\neq \pm 1$, declare n probably prime.

Miller-Rabin primality test

boolean definitelyComposite(a, n)

Compute s and m s.t. $n - 1 = 2^s \cdot m$ and m is odd

$x = a^m \bmod n$

if $x = 1$ return false

for $j = 0$ to $s - 1$

if $x = n - 1 \in \mathbb{Z}_n^*$ return false

$x = x^2 \bmod n$

if $x = 1$ return true

return true

boolean probablyPrime(n)

for $k = 1$ to t

choose a at random from $\{2, 3, \dots, n - 1\}$.

if definitelyComposite(a, n) return false

return true

Factoring methods

Elementary methods such as trial division are useless for factoring cryptographically interesting numbers. (Still useful for small numbers, in the range $1 < N < 10^{12}$, say.)

For big numbers, advanced methods are used. Currently best method, number field sieve, has complexity $L_N(1/3, c)$, where

$$L_N(\alpha, \beta) = \exp((\beta + o(1))(\log N)^\alpha (\log \log N)^{1-\alpha}).$$

This, Moore's law and actual factoring results lead to current recommendations of key size 1024-2048 bits for RSA.

A basic idea

Let $N = pq$, where p and q are primes.

To factor N , try to find x and y , such that $x \neq \pm y \bmod N$ and $x^2 = y^2 \bmod N$.

Then

$$(x^2 - y^2) \bmod N = (x - y)(x + y) \bmod N = 0.$$

It follows that p is a factor of $x - y$ and q a factor of $x + y$ (or conversely) and thus we can find p or q by computing $\gcd(x - y, N)$.

Modern techniques for factoring

Fix a bound B and let p_1, p_2, \dots, p_k be all primes $< B$.

Use more or less clever techniques to find many relations of the form

$$p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_k^{e_k} = z^2 \pmod{p}.$$

Example

Let $N = 221$ and $B = 8$. By trial and error we find

$$2^3 \cdot 3 \cdot 7 = 44^2 \pmod{221}$$

$$2 \cdot 3 \cdot 7 = 56^2 \pmod{221}$$

$$5 \cdot 7 = 120^2 \pmod{221}$$

$$2 \cdot 3 \cdot 5 = 145^2 \pmod{221}$$

$$2^3 \cdot 3 \cdot 5 = 186^2 \pmod{221}$$

How can we make use of these identities?

Cont'd

Multiplying the two last identities gives

$$2^4 \cdot 3^2 \cdot 5^2 = 145^2 \cdot 186^2 \pmod{221}$$

$$(2^2 \cdot 3 \cdot 5)^2 = (145 \cdot 186)^2 \pmod{221}$$

$$60^2 = 8^2 \pmod{221}.$$

Thus we are lead to compute $\gcd(60 - 8, 221) = 13$, and we have found a factor of N .

Which is the general technique to find which equations to multiply?

Cont'd

Let $x_i = 1$ if relation i is used, 0 otherwise. Then the power of 2 in the LHS of the resulting equation is

$$3x_1 + x_2 + 0x_3 + x_4 + 3x_5.$$

We want this to be even. Thus it is enough to work modulo 2 and get the equation

$$x_1 + x_2 + 0x_3 + x_4 + x_5 = 0$$

which we want to solve in \mathbb{Z}_2 .

We get similar equations for powers of 3, 5 and 7, leading to a linear system of equations with 4 equations in 5 unknowns.

This can be solved using standard linear algebra techniques, which work in \mathbb{Z}_2 .

Factoring: summary

To factor N :

- Choose a suitable bound B . For large N , B may be in the order of millions.
- Find slightly more relations than the number of primes $< B$. This step may be parallelized (on a large scale, over the Internet). Different algorithms use different techniques here.
- Solve the resulting equation system (which may have hundreds of thousands of unknowns and equations). This requires special techniques to handle the large amounts of data.

The Chinese Remainder Theorem (CRT)

We will consider only a special case of this ancient theorem (Sun Zi, ca 300 AD).

Theorem (Chinese Remainder Theorem)

Let p and q be distinct primes and $N = p \cdot q$.

For any pair $(x_1, x_2) \in \mathbb{Z}_p \times \mathbb{Z}_q$, there is a unique number $x \in \mathbb{Z}_N$ such that

$$x_1 = x \pmod{p},$$

$$x_2 = x \pmod{q}.$$

CRT representation

Example

Let $N = pq = 3 \cdot 5$.

x	(x_1, x_2)	x	(x_1, x_2)
0	(0, 0)	8	(2, 3)
1	(1, 1)	9	(0, 4)
2	(2, 2)	10	(1, 0)
3	(0, 3)	11	(2, 1)
4	(1, 4)	12	(0, 2)
5	(2, 0)	13	(1, 3)
6	(0, 1)	14	(2, 4)
7	(1, 2)		

CRT representation, better

Example

More instructive to sort by (x_1, x_2) .

Let $N = pq = 3 \cdot 5$.

(x_1, x_2)	x	(x_1, x_2)	x	(x_1, x_2)	x
(0, 0)	0	(1, 0)	10	(2, 0)	5
(0, 1)	6	(1, 1)	1	(2, 1)	11
(0, 2)	12	(1, 2)	7	(2, 2)	2
(0, 3)	3	(1, 3)	13	(2, 3)	8
(0, 4)	9	(1, 4)	4	(2, 4)	14

Arithmetic in the CRT representation

Let x and $y \in \mathbb{Z}_N$ have CRT representations (x_1, x_2) and (y_1, y_2) , respectively.

Let e_{CRT} denote the CRT representation of e .

It is easy to check that

$$(x + y)_{CRT} = ((x_1 + y_1) \pmod{p}, (x_2 + y_2) \pmod{q})$$

$$(x - y)_{CRT} = ((x_1 - y_1) \pmod{p}, (x_2 - y_2) \pmod{q})$$

$$(x \cdot y)_{CRT} = ((x_1 \cdot y_1) \pmod{p}, (x_2 \cdot y_2) \pmod{q})$$

So, to compute in \mathbb{Z}_N , we can convert to the CRT representation and compute there. But

- how can we convert the result back to \mathbb{Z}_N ?
- is this more efficient than computing in \mathbb{Z}_N ?

Going back: Garner's formula

We want to recover $x \in \mathbb{Z}_N$ from its CRT representation (x_1, x_2) .

We have $x = x_1 + u \cdot p$, where
 $u = ((x_2 - x_1)(p^{-1} \bmod q)) \bmod q$.

To prove this, just compute CRT representation of the claimed x and check that you get (x_1, x_2) .

Note also that $p^{-1} \bmod q$ can be computed once and for all for a given N .

RSA and CRT

CRT representation is even more useful for modular exponentiation:

$$(x^s \bmod N)_{CRT} = (x_1^s \bmod p, x_2^s \bmod q) = (x_1^{s \bmod (p-1)}, x_2^{s \bmod (q-1)}).$$

Recall that modular exponentiation is $O(K^3)$, where K is the size in bits of the numbers involved.

So, if p and q are of the same size, each of the exponentiations to the right is 2^3 times faster than the original one, for a total efficiency gain of a factor 4.

The discrete logarithm problem

We recall the discrete log problem, which in the original setting was:

Given a prime p , a generator g for \mathbb{Z}_p^* and $y = g^x \in \mathbb{Z}_p^*$, find x .

In a more general situation, we can consider a g that generates just a (large) subgroup of \mathbb{Z}_p^* , or, even more generally, any cyclic group.

The discrete log problem in cyclic groups

- A cyclic group of order n is a tuple $(G, *, 1, g)$, where
 - G is a finite set with n elements,
 - $*$ is an associative binary operation on G ,
 - 1 is a unit, i.e. $1 * a = a * 1 = a$ for all $a \in G$.
 - $g \in G$ has order n .
- We use exponentiation notation a^x to denote $a * a * a * \dots * a$, even though the operation $*$ need not be multiplication.
- The discrete log problem is, still, to find $x \in \mathbb{Z}_n^+$, given $y = g^x$.

Discrete logs 1: meet in the middle

Baby step/Giant step algorithm

Define $n_0 = \lceil \sqrt{n} \rceil$. We know that $x = a + b \cdot n_0$, where both a and b are less than n_0 .

(Here $\lceil e \rceil$ denotes the ceiling of e , i.e. the smallest integer greater than or equal to e .)

- Tabulate (g^a, a) , for $a = 0, 1, \dots, n_0 - 1$.
- Compute y/g^{bn_0} for $b = 0, 1, \dots, n_0 - 1$ and look up in the table. When a match occurs, $x = a + bn_0$.

Complexity of algorithm

The Baby step/Giant step method requires $O(\sqrt{n})$ steps and $O(\sqrt{n})$ space. The latter is typically more of a problem.

CHALMERS

Square-root time algorithms

Sketch of Pollard's ρ algorithm, part 1

Solves discrete log problem in expected time $O(\sqrt{n})$ using little space.

- Generate "random-looking" sequences of integers a_k and b_k , where $(a_{k+1}, b_{k+1}) = f(a_k, b_k)$ for some function f and compute $t_k = g^{a_k} y^{b_k}$, $k = 0, 1, 2, \dots$
- Within $O(\sqrt{n})$ steps, we will expect a collision $t_i = t_{i+j}$ for some i and j (birthday paradox).
- Then we will have $a_i + b_i x = a_{i+j} + b_{i+j} x \pmod{n}$, hence $x = (a_j - a_{i+j}) / (b_i - b_{i+j}) \in \mathbb{Z}_n^*$.

So far, expected time is OK, but we still need $O(\sqrt{n})$ space to record t_i values in order to discover collisions.

CHALMERS

Cont'd

Sketch of Pollard's ρ algorithm, part 2

To minimize space usage, use **Floyd's cycle-finding algorithm**:

Compute only the sequence (t_k, t_{2k}) , $k = 0, 1, 2, \dots$ without storing more than the current pair. In expected time $O(\sqrt{n})$, we have a **collision in the pair**. Why?

Since $t_i = t_{i+j}$, we have $t_k = t_{k+j}$ for all $k \geq i$ (same function applied to the exponents on both sides).

Now choose k such that $k > i$ and k is a multiple of the cycle length j . Then $t_k = t_{2k}$.

Omitted from the sketch: how to choose f to get "random-looking" behaviour and efficient computation of successive t_k .

CHALMERS

Discrete logs 2: Pohlig-Hellman

We consider a cyclic group with generator g and order $n = q \cdot r$, where primes q and r are known.

We seek x , an integer in the range $0 \leq x < n$.

We attack the problem by finding the CRT representation (x_1, x_2) of x , i. e. $x_1 = x \pmod{q}$ and $x_2 = x \pmod{r}$. We can then find x by Garner's formula.

First we find x_1 .

We note that $g_1 = g^r$ generates a subgroup of order q : g_1^k for $k = 0, 1, \dots, q-1$ are all different and $g_1^q = 1$.

CHALMERS

Cont'd

We raise both sides of $y = g^x$ to power r :

$$y^r = (g^x)^r = g_1^x = g_1^{x_1+kq} = g_1^{x_1} (g_1^q)^k = g_1^{x_1}.$$

Thus x_1 can be found by solving another discrete log problem, now in a subgroup of size q : find x_1 when $g_1^{x_1}$ is known.

Similarly, we find x mod r by solving a discrete log problem in the cyclic group of order r generated by g^q .

This algorithm extends to more than two (repeated) prime factors.

CHALMERS

Cont'd

Example

If q and r are roughly of the same size, and the subproblems are solved by a Pollard type method, total time is $O(n^{1/4})$.

Conclusion

Almost all the work is done in solving the two subproblems.

To protect against this attack, group order must have a large prime factor.

CHALMERS

Discrete logs 3: Index calculus

Applies only to discrete log problem in special groups, such as (subgroups of) \mathbb{Z}_p^* with generator g . We only sketch the idea.

Fix a bound B and let p_1, p_2, \dots, p_k be the primes $< B$.

Use clever techniques to find at least k relations of the form

$$p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_k^{e_k} = g^z \pmod{p}.$$

Such a relation can equivalently be written

$$e_1 x_1 + e_2 x_2 + \dots + e_k x_k = z \pmod{p-1}$$

where $x_j = \log_g(p_j)$. This leads to a system of linear equations to be solved modulo $p-1$, giving all the x_j .

The above is done once for each (p, g) combination.

Cont'd

To solve $y = g^x \pmod{p}$, use clever techniques to write

$$y = p_1^{c_1} \cdot p_2^{c_2} \cdot \dots \cdot p_k^{c_k}.$$

Then, again looking at exponents,

$$x = c_1 x_1 + c_2 x_2 + \dots + c_k x_k \pmod{p-1}.$$

The similarity with factoring methods may hint to why recommended size for p is similar, 1024 bits.

CHALMERS

Discrete logs, summary

The attacks we have seen explain the recommended setup for DSA, ElGamal, Diffie-Hellman and other systems based on discrete logarithms.

- Pohlig-Hellman implies that group size must have large prime factor.
- Baby step/Giant step and Pollard type methods imply that this prime factor should be in the order of 2^{160} for a security level of 80 bits.
- Index calculus implies that if subgroup of \mathbb{Z}_p^* is chosen, size of p (and hence size of numbers involved) should be in the order of 1024 bits.