About these slides

- Part of the Cryptanalysis course I taught at UCL in 2015 for the Master in Information Security
- Contain background computer algebra algorithms useful for both that course and this one
- The slides will not be covered during this course
- Best usage : know what is in them and consult when needed

Public Key Cryptanalysis

Algorithmic Number Theory Basics

Christophe Petit

University College London

Secure communications

- Alice wants to send a **private** message to Bob over a **public** channel
- Private key cryptography : Alice and Bob both have a key to some encryption box

- Public key cryptography : Alice uses a lock of which only Bob has the key

Public key vs Private key cryptography

- No preshared password needed with public key crypto
- Security *reduced* to "hard" number theory problems vs. "ad hoc" security for block ciphers, hash functions
- Mathematical problems have independent interest, so more scrutinized. . . for the best and the worst
- Typically $\sim$ 1500 bits vs. $\sim$ 160 bits

Module objectives

- Revise algorithmic number theory basics from IntroCrypto
- Revise Linear Algebra basics
- If time : learn root-finding algorithms
- Lab & tutorial : discover SAGE and connect theory to practice, play with some first attacks

Outline

Complexity measures

Algebra and number theory

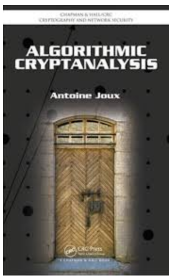
First algorithmic number theory tools

Linear algebra

Root-finding algorithms

Reference book

- Algorithmic Cryptanalysis, Chapters 1-3



Outline

Complexity measures

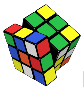
Algebra and number theory

First algorithmic number theory tools

Linear algebra

Root-finding algorithms

What do we mean by "hard" problem ?

- Is  hard ?
- Is adding two integers hard ?
- Is multiplying two integers hard ?
- Is factoring integers hard ? what about 15 ?
- Is inverting a matrix hard ? what if it has billions of rows and columns ?

Big Oh notation

- Let $f, g : \mathbb{N} \to \mathbb{R}$. We say $f = O(g)$ if there exist N and c such that for all $n > N$, we have $g(n) \geq cf(n)$.
- Examples :
  - $x = O(x^2)$
  - $10000000x = O(x^2)$
  - $x^n = O(e^x)$ for any n
  - $\log x = O(x)$

Measuring complexity (theory)

- Consider the multiplication problem :
  given two integers p and q, compute $n = pq$
- Hardness is function of $s := \log_2 p + \log_2 q$, the input size
- Trivial algorithm runs in time $O(\log_2 p \cdot \log_2 q) = O(s^2)$: multiply p by each bit of q, shift by appropriate powers of 2, and make additions with carries
- Best algorithms achieve $O(s \log s)$

Measuring complexity (theory)

- Consider exhaustive search on a key of n bits
- Hardness is function of n
- Complexity is $O(2^n)$: try every possible key
- Exponential complexity !

Measuring complexity (theory)

- Consider the factorization problem : given a positive composite integer n, find p and q such that $n = pq$
- Hardness is function of $\log_2 n$, that is the size of input
- The best algorithms today run in **subexponential** time

$$L_n(\alpha; c) = \exp(c(\log n)^{\alpha}(\log\log n)^{1-\alpha})$$

with $\alpha = 1/3$

P and NP

- A problem is in P if it can be solved in polynomial time (in other words, there is an integer n such that it can be solved in time $O(x^n)$ for an input of size x)
- Refinements to this : randomization, memory, etc.
- A problem is in NP if a solution can be checked in polynomial time
- P=NP ? is worth a million dollards (and glory !)
- NP-complete problems are as hard as the hardest known NP problems such as 3-SAT, graph coloring, traveling salesman, etc
- Factorization, Dlog, are (probably) NOT NP-complete

In practice

- Hardness depends on your computer power, your time, your memory
- Hard for you might be easy for NSA
- Compare with exhaustive search : 2^{20} is certainly possible on a laptop, 2^{60} becomes very hard for most organizations
- See `www.keylength.com` for key sizes

Outline

Group

- A group $(G, \circ)$ is a set G with some binary operation $\circ : G \times G \to G$ such that
  - Neutral element : there exists $e \in G$ such that for all $x \in G$, we have $x \circ e = x = e \circ x$
  - Inverse : for all $x \in G$, there exists y such that $x \circ y = e = y \circ x$
  - Associativity : for all $x, y, z \in G$, we have $(x \circ y) \circ z = x \circ (y \circ z)$
- When $\circ$ is implicit, we say G is a group
- A group is Abelian if for all x, y, we have $x \circ y = y \circ x$
- A group is finite if $|G|$ is finite

Group examples

- $(\mathbb{Z}, +)$ is a group with neutral element 0
- $(\mathbb{Q}, +)$ is a group with neutral element 0
- $(\mathbb{Q}, *)$ is not a group : 0 has no inverse
- $(\mathbb{Q}^*, *)$ is a group with neutral element 1
  Here $\mathbb{Q}^* = \mathbb{Q} \setminus \{0\}$
- $(\mathbb{Z}_n, +)$ is a group for any positive integer n
  Here $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$ are integers modulo n
- $(\mathbb{Z}_p^*, *)$ is a group for any prime number p
  Here $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$
- ...

Rank of a group

- The rank of a group $(G, +)$ is the minimal number of elements needed to generate the whole group

$$\min\{k : \exists S = \{g_1, \ldots, g_k\} \subset G \text{ s.t. } \forall g \in G, g = \sum_i g_{e_i} \text{ with } g_{e_i} \in S\}$$

- Example : $(\mathbb{Z} \times \mathbb{Z}, +)$ is a group of rank 2 with generating set $\{(1,0),(0,1)\}$
- A group of rank 1 is called a cyclic group

Lagrange theorem

- Let $(G, \circ)$ a finite group
- For any integer k and any $g \in G$, we write g^k for $g \circ g \circ \ldots \circ g$, k times
- Lagrange's theorem : for any $g \in G$, we have $g^{|G|} = e$ where e is the neutral element in the group
- Fermat's small theorem : for any prime p and any $g \neq 0 \bmod p$, we have $g^{p-1} = 1 \bmod p$

Field

- A field $(K, +, *)$ is a set K with two binary operations $+ : K \times K \to K$ and $* : K \times K \to K$ such that
  - $(K, +)$ is an Abelian group
  - $(K^*, *)$ is a group, where $K^* = K \setminus \{e\}$, where e is the neutral element of K for $+$
- A field $(K, +, *)$ is finite if $|K|$ is finite

Field examples

- $(\mathbb{C}, +, *)$ is a field with neutral elements 0 and 1 for $+$ and *
- $(\mathbb{Q}, +, *)$ is a field with neutral elements 0 and 1 for $+$ and *
- $(\mathbb{Z}_p, +, *)$ is a finite field for any prime p
  This field is often denoted $\mathbb{F}_p$

A more complicated example

- Let f be a polynomial of degree n with coefficients in $\mathbb{F}_p$, such that f has no factor of degree different than 0 or n.
- Consider $(K, +, *)$ where
  - K = all polynomials over $\mathbb{F}_p$
  - $+$ and $*$ are addition and multiplication modulo the polynomial f
- Then $(K, +, *)$ is a finite field with p^n elements
- Example : let $f(x) = x^2 + x + 1 \in \mathbb{F}_2[x]$ then $\mathbb{F}_4 = \mathbb{F}_2[x]/(f(x)\mathbb{F}_2[x])$ is a finite field with 4 elements $\{0, 1, x, x + 1\}$

Vector space

- A vector space $(V, +, *)$ over some field K is a set $V \supset K$ with two operations $+ : V \times V \to V$ and $* : K \times V \to V$ such that
  - $(V, +)$ is a group
  - For all $a, b \in K$ and all $v \in V$, we have $(a + b) * v = a * v + b * v$
  - For all $a \in K$ and $v, w \in V$, we have $a * (v + w) = a * v + a * w$
- The dimension of this vector space is the rank of $(V, +)$
- A basis of V is a set of $(\dim V)$ elements that generate V

Ring

- A ring $(R, +, *)$ is a set R with two operations $+ : R \times R \to R$ and $* : R \times R \to R$ such that
  - $(R, +)$ is an Abelian group
  - $(R, *)$ is associative and has a neutral element (but some elements may have no inverse)
  - Distributivity : for all $a, b, c \in R$, we have $(a + b) * c = a * c + b * c$

Ring examples

- Let K be a field and let $K[X]$ be the set of polynomials with coefficients in K. Then $(K[X], +, *)$ is a ring
- $\mathbb{Z}_n := \mathbb{Z}/n\mathbb{Z}$ (the integers modulo n) is a ring for any $n \in \mathbb{N}$. It is a field if and only if n is prime.
- Let K be a field. Let $f \in K[X]$ and let $\tilde{K} = K[X]/(f(X))$ be the set of polynomials over K "modulo $f(x)$".
  Then $\tilde{K}$ is a ring. It is a field if and only if f is irreducible.

Prime numbers

- 2,3,5,7,11,... are prime numbers. 4,6,8,9,10,... are not
- Any integer n can be decomposed uniquely has a product of prime numbers
- There are infinitely many primes
- **Prime number theorem** : the number of primes up to some bound B is roughly equal to $B/\log B$

The RSA ring

- Let p, q be two primes and let $n = pq$
- Let $\mathbb{Z}_n := \mathbb{Z}/n\mathbb{Z}$ be the ring of integers modulo n
- Not a field : for any k, neither kp nor kq are invertible
- The map

$$\varphi : \mathbb{Z}_n \to \mathbb{Z}_p \times \mathbb{Z}_q : x \to (x \bmod p, x \bmod q)$$

is a ring isomorphism. Its inverse is given by

$$\begin{aligned} \varphi^{-1} : \mathbb{Z}_p \times \mathbb{Z}_q &\to \mathbb{Z}_n \\ (x_p, x_q) &\to x_p q (q^{-1} \bmod p) + x_q p (p^{-1} \bmod q) \end{aligned}$$

Chinese remainder theorem

- More generally if $n = \prod_{i=1}^{N} p_i^{e_i}$ then the map

$$\varphi : \mathbb{Z}_n \to \prod_{i=1}^{N} \mathbb{Z}_{p_i^{e_i}} : x \to (x \bmod p_1^{e_1}, \ldots, x \bmod p_N^{e_N})$$

  is a ring isomorphism
- In other words given all residue values, there exists a unique value that corresponds to them modulo n

Euler's theorem

- Let $n = \prod_{i=1}^{N} p_i^{e_i}$ where the p_i are distinct primes
- Define the Euler totient function

$$\varphi(n) = \prod_{i=1}^{N} (p_i - 1) p_i^{e_i - 1}$$

- Then for all $x \in \mathbb{Z}_n^*$, we have

$$x^{\varphi(n)} = 1 \bmod n$$

- If $n = p$ a prime, then $\varphi(n) = p - 1$ and we recover Fermat's small theorem $x^{p-1} = 1 \bmod p$
- If $n = pq$ like in RSA, then $\varphi(n) = (p-1)(q-1)$

Outline

Addition in $\mathbb{F}_p$

- Let p be a prime and let $K := \mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$
- Addition in K : given a and b, return $a + b \bmod p$

```
c ← a + b
if c > p then
    c ← c − p
end if
return c
```

- Complexity $O(\log p)$ bit operations

Multiplication in $\mathbb{F}_p$

- Let p be a prime and let $K := \mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$
- Multiplication in K : given a and b, return ab mod p

  Let $b = \sum_{i=0}^{n} b_i 2^i$
  $a' \leftarrow a$; $c \leftarrow b_0 a$
  for i=1 **to** n **do**
      $a' \leftarrow 2a'$ mod p
      $c \leftarrow c + b_i a'$ mod p
  end for
  return c

- Complexity $O(n^2) = O(\log^2 p)$ bit operations
- Best algorithms achieve $O(\log p \log\log p)$

Modular exponentiation : Square-and-Multiply

- Let p be a prime and let $K := \mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$
- Exponentiation in K : given a and k, return a^k mod p

  Let $k = \sum_{i=0}^{n} k_i 2^i$
  $a' \leftarrow a$; $c \leftarrow a^{k_0}$
  for i=1 **to** n **do**
      $a' \leftarrow a'^2$ mod p
      $c \leftarrow c(a')^{k_i}$ mod p
  end for
  return c

- Complexity $O(n) = O(\log p)$ multiplications

Remark on elementary operations

- The above algorithms can be generalized to a great extent to other fields, ring or group structures, with similar complexities

The discrete logarithm problem

- Let p be a prime and let $K := \mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$
- Exponentiation in K in $O(n) = O(\log p)$ multiplications
- What about the inverse operation ?
- **Discrete logarithm problem :**
Given g **and** $h = g^k$ mod p**, compute** k
- Believed to be very hard : subexponential complexity $L_p(1/3, c)$
- More generally : given G, $g \in G$ and $h = g^k$, compute k
- Can be harder or easier depending on the group

Diffie-Hellman algorithm


- Designed by Diffie and Hellman in 1976
- Widely used today, e.g. in SSL
- Allows two parties to set up a common private key over a public channel
- Security requires hardness of discrete logarithm problem

Diffie-Hellman algorithm

- Public elements : G cyclic, $g \in G$ a generator
- Alice chooses random a and sends g^a to Bob
- Bob chooses random b and sends g^b to Alice
- Alice computes $(g^b)^a = g^{ab}$
- Bob computes $(g^a)^b = g^{ab}$

Diffie-Hellman security

- Solving discrete logarithm problem is sufficient to break Diffie-Hellman key exchange
- Solving discrete logarithm problem might not be necessary to break Diffie-Hellman key exchange
- Additional stuff is required for authentication, for example certificates

Primality testing

- Given an integer n, decide whether n is prime or not
- You can generate primes by picking random numbers smaller than B and checking whether they are prime : need about $\log B$ trials by the prime number theorem
- There are deterministic algorithms for primality testing (see AKS test)
- In practice, we use probabilistic algorithms (having a small probability to return prime for composite numbers) that are much faster

Fermat test

- Observation : if n is prime then $a^n = a \bmod n$ for all a (Fermat's small theorem)
- Idea : choose random a and check whether $a^n = a \bmod n$. If not then p is composite.
- Bad news : some numbers (Carmichael numbers) are composite and satisfy this equation for all $0 < a < n$!

Miller-Rabin test

- Observation : if n is prime, than the only x such that $x^2 = 1 \bmod n$ are $\pm 1 \bmod n$ whereas if n is composite, there are more of them
- Idea : write $n - 1 = 2^k q$, pick random a and compute $a_0 = a^q \bmod n$, then $a_i = a_{i-1}^2 \bmod n$, etc, up to $a_k = a^{n-1} \bmod n$
  - If n is prime : the sequence $(a_0, a_1, \ldots, a_k)$ will be $(*, *, \ldots, *, -1, 1, \ldots, 1)$ where $* \neq \pm 1$
  - If n is composite then it will be $(*, *, \ldots, *, *, 1, \ldots, 1)$ for at least $3/4$ of the values a
- Complexity $O(-\log \epsilon)$ modular exponentiations, where ϵ is error probability

RSA algorithm



- Designed by Rivest-Shamir-Adleman in 1977
- One of the most widely used algorithms today, for both signatures and public key encryption
- Security requires hardness of integer factorization

RSA encryption algorithm

- Let p, q two distinct odd primes, and let $n = pq$
- Let e with no common divisor with $\varphi(n) = (p-1)(q-1)$
- **Public key** is (n, e) and **private key** is (p, q)
- Given private key, can also compute $d := e^{-1} \bmod \varphi(n)$
- Encryption of m is $c = m^e \bmod n$
- Decryption of c is $m' = c^d \bmod n$
- Correctness follows from

$$m' = (m^e)^d = m^{ed \bmod \varphi(n)} = mm^{(ed-1) \bmod \varphi(n)} = m \bmod n$$

by Euler's theorem

RSA security

- Solving the factorization problem is sufficient and necessary to reconstruct the private key
- Solving the factorization problem *might not be necessary* for other goals, such as decrypting without the private key
- In fact, "textbook RSA" insecure wrt some goals : for example given an encryption of m, can compute an encryption of $m^2 \bmod n$

RSA weak key generator attack

- Suppose Alice uses private key (p, q_a) and Bob uses private key (p, q_b). Is it safe ?
- Everybody sees $n_a := pq_a$ and $n_b := pq_b$
- Alice can compute $q_b = n_b/p$
- Bob can compute $q_a = n_a/p$
- **Anyone** can compute $\gcd(n_a, n_b) = p$ and then q_a and q_b
- Attack demonstrated in practice

  Lenstra et al. *Ron was wrong, Whit is right*
  Show that 2/1000 RSA keys are insecure

Euclide algorithm

- Goal : given integers a and b, find $d = \gcd(a, b)$
- $d|a$, $d|b$ imply $d|(a + kb)$ for any integer k

Require: $a \geq b$
Ensure: $\gcd(a, b)$

```
if b|a then
    return b
else
    Compute q such that 0 < a − qb < b
    return gcd(b, a − qb)
end if
```

- Complexity $O(|a|^2)$; best algorithms achieve $O(|a| \log |a|)$

Example

$$\begin{aligned} \gcd(36, 16) &= \gcd(16, 36 - 2 \cdot 32) \\ &= \gcd(16, 4) \\ &= 4 \end{aligned}$$

Extended Euclide algorithm

- Goal : compute r and s such that $ra + sb = \gcd(a, b)$

Require: $a \geq b$
Ensure: $d = \gcd(a, b)$ **and** r, s, such that $ar + bs = d$

```
if b|a then
    return a, 0, 1
else
    Compute q such that 0 < a − qb < b
    d, r, s ← gcd(b, a − qb)
    return d, s, r − qs
end if
```

- Indeed if $rb + s(a - qb) = d$ then $sa + (r - qb)b = d$
- Complexity $O(|a|^2)$; best algorithms achieve $O(|a| \log |a|)$

Example

$$\begin{aligned}\gcd(36,16) &= \gcd(16, 36 - 2\cdot 32)\\ &= \gcd(16,4)\\ &= 4\end{aligned}$$

$$\begin{aligned}4 &= 0\cdot 16 + 1\cdot 4\\ &= 1\cdot 36 + (0 - 2\cdot 1)16\end{aligned}$$

Outline

Complexity measures

Algebra and number theory

First algorithmic number theory tools

Linear algebra

Root-finding algorithms

Main goals in linear algebra

- Multiply two square matrices
- Inverse a square invertible matrix
- Solve linear systems of equations

Complexity of linear algebra

- All these tasks have roughly the same complexity
- For an $n \times n$ matrix, complexity $O(n^\omega)$ multiplications where
  - Lower bound $\omega \geq 2$
  - Gauss elimination $\omega \leq 3$
  - Strassen $\omega \leq \log_2 7 \approx 2.8074$
  - In 2015 we know $\omega \leq 2.3728639$ (but not practical)
  - Conjecture : for any $\epsilon > 0$, we could have $\omega = 2 + \epsilon$
  - ω may be smaller for specific matrices

Scalar product

- Given two vectors $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$, compute their scalar product $c = (a, b) = \sum_{i=1}^n a_i b_i$
- Complexity : n multiplications

Matrix multiplication

- Given two $n \times n$ matrices A and B compute $C = AB$
- See A and B as row and column matrices respectively

$$A = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \qquad B = \begin{pmatrix} b_1 & \ldots & b_n \end{pmatrix}$$

- n^2 scalar products (a_i, b_j), so n^3 multiplications in total

Strassen algorithm

- Idea : trade some multiplications for additions
- Compute a product of $2n \times 2n$ matrices using 7 (instead of 8) products of $n \times n$ matrices
- To compute MM' where $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and $M = \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix}$, compute

$$P_1 = (a+c)(a'+b'), \quad P_2 = (b+d)(c'+d'), \quad P_3 = (b+c)(c'-b')$$
$$P_4 = c(a'+c'), \quad P_5 = b(b'+d'), \quad P_6 = (c-d)c', \quad P_7 = (a-b)b'$$
$$M \cdot M' = \begin{pmatrix} P_1+P_3-P_4-P_7 & P_5+P_7 \\ P_4-P_6 & P_2-P_3-P_5+P_6 \end{pmatrix}$$

- Complexity :

$$T(2n) = 7 \cdot T(n) + O(n^2) \quad \Longrightarrow \quad T(n) = n^{\log_2 7} = n^{2.807}$$

Best asymptotic algorithms

- Coppersmith-Winograd $\omega < 2.375477$
- Between 2010 and 2014 : ω decreased to 2.3728639
- Those fast asymptotic algorithms are not used in practice because of large constants involved
- Conjecture : $\omega = 2 + \epsilon$

From inversion to multiplication

$$D := \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix} \quad \Rightarrow \quad D^{-1} = \begin{pmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{pmatrix}$$

- If inversion takes $O(n^\omega)$ then so does multiplication

From multiplication to inversion

- Given $M = \left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right)$ to invert, let $N := \left(\begin{smallmatrix} I & 0 \\ -D^{-1}C & I \end{smallmatrix}\right)$
- We have $MN = \left(\begin{smallmatrix} S & B \\ 0 & D \end{smallmatrix}\right)$ for $S := A - BD^{-1}C$
- We have $(MN)^{-1} = \left(\begin{smallmatrix} S^{-1} & -S^{-1}BD^{-1} \\ 0 & D^{-1} \end{smallmatrix}\right)$
- Compute D^{-1} then $-D^{-1}C$
- Compute S then S^{-1} then $-S^{-1}BD^{-1}$
- Compute $M^{-1} = N(MN)^{-1}$
- Cost :

$$T_{inv}(2n) = 2T_{inv}(n) + 8T_{mul}(n) + O(n^2)$$

- If multiplication takes $O(n^\omega)$ then so does inversion

Gaussian elimination

- Observation : if $My = x$ then for any invertible N, we have $NMy = Nx$
- In particular, this is true when N is a matrix which
  - Swaps two rows of M
  - Multiplies one row by an invertible constant
  - Adds a multiple of one row of M to another row of M
- Gaussian elimination repeats these operations until the resulting matrix is upper triangular

Gaussian elimination

- Algorithm when M is invertible
  1: **for** each column i, from $i = 1$ **to** n **do**
  2: Find a nonzero element in this column
  3: Swap the row of this element with row i
  4: **for** each row j below row i **do**
  5: Let $c := -M_{j,i}/M_{i,i}$
  6: Add c times row i to row j to erase the value in (j, i)
  7: **end for**
  8: **end for**
- Adapt step 2 otherwise
- Cost is $O(n^3)$ multiplications

Resolution from Gaussian form

- Algorithm when M is invertible
  1: **for** each column i from n to 1 **do**
  2: Recover value of unknown i, using equation i and all values of previously computed unknowns $j > i$
  3: **end for**
- Adapt to determine the afine space of solutions $v + \ker M$ otherwise
- Cost is $O(n^2)$ multiplications
- Can be used to invert M in $O(n^3)$ multiplications

Hermite normal form

- If the matrices are defined over a ring (not a field) then not all elements are invertible
- Elimination in each column will be done with a kind of GCD algorithm :
  1: **for** each column i, with $i \in \{0, \ldots, n\}$ **do**
  2: **while** some element below (i, i) is non zero **do**
  3: Find the smallest nonzero element in this column
  4: Swap the row of this element with row i
  5: For each row j below i, remove as many times row i as needed to have element (j, i) between 0 and element (i, i)
  6: **end while**
  7: **end for**

Hermite normal form

- A matrix is in Hermite normal form if it is upper triangular, has positive elements on the diagonal, and moreover all non-diagonal elements are non-negative and smaller than the diagonal elements in their column
- Last condition ensured by completing previous algorithm with
  1: **for** each column i, with i from 1 to n **do**
  2: For each row j above i, remove as many times row i as needed to have element (j, i) larger than 0 and smaller than element (i, i)
  3: **end for**

Sparse linear algebra

- A matrix is **sparse** if each row contains a small number of nonzero elements
- Very useful in index calculus algorithms (see topic 2) and many other contexts
- Can store larger size matrices by storing only $(i, j, M_{i,j})$ for nonzero elements $M_{i,j}$
- Gaussian elimination will kill the sparsity quickly
- Two approaches for sparse matrices :
  - Structured Gaussian elimination
  - Algorithms based on matrix-vector multiplications

Structured Gaussian elimination

- Consider the linear system $My = x$
- For the matrices M occurring in index calculus :
  - Each row contain few elements
  - The first columns contain much more elements than the last ones
- Structured Gaussian elimination involves several tricks such as removing variables that only appear once or twice
- Used as preprocessing ro reduce the size in practice
- Heuristic

Lanczos algorithm

- If M is invertible, $My = x \Leftrightarrow M^t My = M^t x$ hence we can assume M is symmetric defining a scalar product $(x, y)_M := xMy^t$
- Lanczos is iterative : over the real/complex numbers, the algorithm can be stopped before the end with a reasonable approximation of the solution
- First compute a basis $\{v_i\}$ of orthogonal vectors with respect to the scalar product $(*, *)_M$ (see topic 3), then compute $x = \sum_{i=1}^{n} (x, v_i)_M v_i$
- First part involves $O(n)$ matrix-vector multiplications, each one at $O(n)$ cost if each row contains $O(1)$ elements

Wiedemann algorithm

- Reconstruct the **minimal polynomial** of M : smallest degree polynomial f such that $f(M) = 0$
- If $f(\alpha) = \sum_{i=0}^{d} f_i \alpha^i$, then $I = -\frac{1}{f_0} \sum_{i=1}^{d} f_i M^i$ then

$$x = -\frac{1}{f_0} \sum_{i=1}^{d} f_i M^i x = M \left(-\frac{1}{f_0} \sum_{i=1}^{d} f_i M^{i-1} x \right)$$

- We deduce y such that $My = x$
- The algorithm requires $O(n)$ matrix-vector products
- Recent discrete log records use Block Wiedemann `http://caramel.loria.fr/p180.txt`

Outline

Root-finding algorithms

- Problem : given a polynomial $f \in \mathbb{F}_q[x]$, find all $x \in \mathbb{F}_q$ such that $f(x) = 0$
- Note that $f(\alpha) = 0 \Leftrightarrow (x - \alpha)|f(x)$
- First compute the **square-free split** part of f : the unique monic polynomial $\tilde{f}|f$ such that $\tilde{f}$ has only linear factors, all distinct
- Then successively split $\tilde{f}$ into smaller polynomials

Square-free split part

- We have $\alpha^q = \alpha$ for all $\alpha \in \mathbb{F}_q$ so
$$x^q - x = \prod_{\alpha \in \mathbb{F}_q} (x - \alpha)$$
- Therefore
$$\tilde{f}(x) = \gcd(x^q - x, f(x))$$
contains only factors of degree 1, no factor twice
- Compute $x^q \bmod f(x)$ with a square-and-multiply algorithm, substract x, and compute gcd

Breaking out $\tilde{f}$

- If q odd we have
$$x^q - x = x(x^{\frac{q-1}{2}} + 1)(x^{\frac{q-1}{2}} - 1)$$
- Computing
$$\gcd\left(\tilde{f}, x^{\frac{q-1}{2}} \pm 1\right)$$
likely to break $\tilde{f}$ into two parts
- Also notice that $x^q - x = (x - a)^q - (x - a)$ for all $a \in \mathbb{F}_q$
- Pick a random a and compute
$$\gcd\left(\tilde{f}, (x - a)^{\frac{q-1}{2}} \pm 1\right)$$

Remarks

- Several other algorithms
- Polynomial time in $\deg f$ and $\log q$
- Can be adapted when q is even
- Can be generalized to find other factors of f, not just degree 1 factors