Slide 1

Clefs publiques et complexité

B. Martin

Slide 2

Cryptologie

Science du chiffre

But : assurer la confidentialité des communications en présence d'adversaires [1].

Cryptologie = cryptographie + cryptanalyse

Cryptographie :
- à clé secrète (depuis l'antiquité)
- à clé publique (depuis 1976)

Cryptanalyse :
- à cryptogramme connu
- à clairs/cryptogrammes connus

Slide 3

Complexité

Taxonomie des problèmes algorithmiques

But : ranger les problèmes algorithmiques en deux classes :

– celle des problèmes *efficaces*
(multiplication, exponentiation modulaire. . .)

– celle des problèmes *algorithmiquement intraitables*
(sac à dos, factorisation, primalité, logarithme discret. . .)

La complexité des problèmes a été introduite par Rabin, McNaughton et Yamada dans les années 1960. Elle a pris corps dans les années 1970 avec Cook et Karp (introduction des notions de problèmes NP -complets et des réductions).

Slide 4

Alphabets, mots, langages

Soit Σ un **alphabet**, (ensemble fini non vide de symboles)
p.e. l'alphabet binaire $\{0, 1\}$; l'alphabet des lettres $\{a, \ldots, z\}$.

Un **mot fini** est une suite finie de symboles sur cet alphabet,
p.e. 01101 sur $\{0, 1\}$, "*complexite*" sur $\{a, \ldots, z\}$.

$\Sigma^\star$ représente l'ensemble des mots finis sur l'alphabet Σ.

Muni d'une opération associative la **concaténation** notée . et d'un **élément neutre** ε, $(\Sigma^\star, .)$ est un **monoïde**.

On définit un **langage** comme un sous-ensemble de $\Sigma^\star$. On a quelques langages particuliers : le langage vide, $L = \varnothing = \{\}$ différent du langage du mot vide $L = \{\varepsilon\}$. Un langage peut être soit fini comme p.e. $L = \{0110, 1001\}$, soit infini dénombrable comme le langage des mots binaires pairs.

Problèmes de décision et codage

La complexité traite essentiellement de **problèmes de décision** définis comme une collection d'**instances** qui sont des ensembles de données et qui admettent deux réponses «oui» ou «non» à une certaine **question**.

INSTANCE : n un entier

Slide 5

QUESTION : n est-il pair ?

On code ensuite les instances positives du problème sous la forme d'un langage ; p.e. le codage binaire du problème précédent est

$$L(B) = 1\{0,1\}^\star 0$$

On utilise ensuite un **algorithme** pour décider si un mot donné en entrée appartient ou non au langage.

Déterminisme et non déterminisme

On distingue deux types d'algorithmes de décision d'un langage L :

Slide 6

– les *déterministes* : étant donnée une instance x, l'algorithme décide si x appartient ou non à L en suivant les instructions de calcul pas à pas.
(exemple : algorithme de décision de la parité décide si 10110 est pair)

– les *non-déterministes* : étant donnée une instance x, l'algorithme va interroger un oracle qui va lui fournir un *certificat* y ; il va ensuite pouvoir décider de manière déterministe si x appartient ou non à L en utilisant le certificat y.
(exemple : 68689 est-il composé ? certificat : 149)

Slide 7

Complexité en temps

Etant donné L, le codage d'un problème de décision Π, la *complexité en temps* d'un algorithme de décision pour L est une fonction $f : \mathbb{N} \to \mathbb{N}$ qui exprime le plus long temps de calcul $f(n)$ nécessaire à la décision d'une entrée de taille $n \in \mathbb{N}$.

On dit que l'algorithme travaille en *temps polynomial* si sa complexité en temps $f(n)$ vérifie pour tout $n \in \mathbb{N}$

$$f(n) \leq p(n)$$

pour un certain polynôme p.

Slide 8

Classes de complexité

On définit la classe du *temps déterministe polynomial* notée P par :

P $= \{L : \exists$ un algorithme déterministe de décision pour L en temps polynomial$\}$

et la classe du *temps non-déterministe polynomial* notée NP par l'ensemble des langages décidables en temps polynomial par un algorithme non déterministe (et dont le certificat est de taille polynomiale).

P $\subseteq$ NP car tout algorithme déterministe est un cas particulier d'un algorithme non déterministe.

Réciproquement, il s'agit de décider si tout langage accepté par un algorithme non déterministe en temps polynomial est également accepté par un algorithme déterministe en temps polynomial.

C'est la célèbre conjecture **P=NP**, un problème à un million de dollars [4] :

`http ://www.claymath.org/prizeproblems/index.htm`

Transformation polynomiale

Définition 1 *Soient $L_1 \subseteq \Sigma_1^\star$ et $L_2 \subseteq \Sigma_2^\star$. Une* ***transformation polynomiale*** *de L_1 vers L_2 (notée $L_1 \propto L_2$) est une fonction $f : \Sigma_1^\star \to \Sigma_2^\star$ tq :*

1. *f est calculable en temps polynomial par un algorithme déterministe ;*
2. *$f(x) \in L_2 \Leftrightarrow x \in L_1$.*

Slide 9

Lemme 1 *$L_1 \propto L_2 \Rightarrow$ (1) si $L_2 \in$ P, alors $L_1 \in$ P ; (2) si $L_1 \notin$ P, alors $L_2 \notin$ P.*

Lemme 2 *Si $L_1 \propto L_2$ et $L_2 \propto L_3$, alors $L_1 \propto L_3$.*

La transformation polynomiale permet de définir les *langages* NP *complets*.

Définition 2 *L est* NP *-complet si $L \in$* NP *et $\forall L' \in$* NP *, $L' \propto L$.*

Prouver la NP -complétude de Π

Il faudrait prouver que tout langage de NP se transforme polynomialement en le langage Π pour montrer que Π est NP -complet.

Simplification :

Lemme 3 *Soient $L_1 \in$* NP *, $L_2 \in$* NP *. Si L_1 est* NP *-complet et $L_1 \propto L_2$, alors L_2 est* NP *-complet.*

Slide 10

Mais il faut connaitre des langages NP -complets ! C'est le théorème de Cook qui a initialisé le processus en utilisant un autre type de réduction.
INSTANCE : U, ensemble de variables et C, collection de clauses sur U

QUESTION : Existe-t'il une valuation satisfiable pour C ?

Théorème 1 (Cook [6]) *SAT est* NP *-complet*

Invention des clés publiques

Slide 11

L'idée de la cryptographie à clé publique est récente.

Elle provient de l'article fondateur de Diffie et Hellman [5], dont la première phrase est presque prophétique :

> *Nous nous trouvons aujourd'hui à l'aube d'une révolution en cryptographie.*

Le principe [1] en est simple : le mécanisme de chiffrement est différent de celui de déchiffrement.

Le chiffrement est fait au moyen d'une clé de chiffrement **publique**,

Le déchiffrement est effectué au moyen d'une clé de déchiffrement **privée**.

Fonction à sens unique à trappe

Slide 12

Une fonction $f : M \to C$ est dite **à sens unique** si

– pour tout x de M, il est **facile** de calculer $f(x)$
(autrement dit, f peut être calculée en temps polynomial) et

– il est **difficile** de trouver, pour la plupart des $y \in f(M)$ un $x \in M$ tel que $f(x) = y$ à moins d'exécuter un nombre prohibitif d'opérations, ou d'avoir une chance sur laquelle il est déraisonnable de compter.

Le calcul dans le sens inverse sera lui aussi efficace pourvu qu'on dispose d'une information secrète -la **trappe**- i.e. une fonction g telle que $g \circ f = Id$.

La construction des couples (f, g) doit être facile et la publication de f ne doit rien révéler sur g.

C'est là qu'on retrouve formalisée l'idée d'utiliser deux algorithmes différents, un pour chiffrer et un pour déchiffrer.

Merkle Hellman–Présentation

Les problèmes NP -complets [6] semblent de bons candidats pour construire des fonctions à sens unique (si on peut trouver une fonction trappe).

Merkle et Hellman on choisi le problème suivant [8] :

Slide 13

Instance : $A = (a_1, \ldots, a_n)$, n-uplet d'entiers distincts et un entier k.
Question : Existe-t-il un sous-ensemble de A dont la somme est égale à k ?

Exemple : $A = (232, 104, 147, 105, 70)$ et $406\ldots$

Fonction à sens unique : tout entier x, $0 \leq x \leq 2^n - 1$ peut être représenté en binaire sur n bits. On définit $f(x)$ comme le produit scalaire $\langle A, \overline{x} \rangle$

Exemple : $f(25) = \langle A, 11001 \rangle = 406$.

Remarque : Pour le moment notre système de chiffrement n'est pas à clé publique. Il lui manque une trappe !

Merkle Hellman–Présentation

On cherche des classes de problèmes de sac à dos **faciles**. C'est le cas si les éléments de A forment une suite appelée *super-croissante* :

$$\forall j, 1 < j \leq n, \sum_{i=1}^{j-1} a_i < a_j$$

Slide 14

Dans ce cas, résoudre le problème est (trop) facile. Il faut perturber le sac à dos supercroissant de telle sorte que le n-uplet résultant ressemble à un problème du sac à dos arbitraire en utilisant une multiplication modulaire.

On choisit un entier $m > \sum_{i=1}^{n} a_i$ et un entier t premier avec m pour disposer de t^{-1} modulo m.

On calcule $b_i \equiv a_i \times t \mod m$ pour tout $i, 1 \leq i \leq n$ et obtient un nouveau n-uplet B : la clé publique.

Exemple : $A = (12, 14, 27, 55, 120)$, $m = 250$, $t = 61$ et $t^{-1} = 41$.
$B = (232, 104, 147, 105, 70)$.

Déchiffrer

Le destinataire connaît les entiers t, t^{-1} et m et le n-uplet A.

Après avoir reçu un bloc chiffré $c' \in \mathbb{N}$, il calcule $t^{-1}c' \equiv c \mod m$ et résout le problème du sac à dos défini par A et c.

Slide 15

La solution définit une suite unique p de n bits. Il s'agit aussi d'un bloc du clair car toute solution p' du problème pour B et c' est égale à p :

$$c \equiv t^{-1}c' \equiv t^{-1}B.p' \equiv t^{-1}tAp' \equiv Ap' \mod m$$

Observons que $Ap' < m$ car $m > \sum_{i=1}^{n} a_i$, ce qui implique que la congruence ci-dessus se simplifie en $c = Ap'$.

Comme le problème défini par A et c ne peut avoir plusieurs solutions, on a nécessairement $p = p'$.

Cryptanalyse par LLL

Définition 3 ([3]) *Un **réseau entier** est un sous-groupe de $\mathbb{Z}^n$, i.e. un ensemble de vecteurs à coordonnées entières qui vérifie :*
– l'opposé de tout vecteur du réseau est dans le réseau ;
– la somme de 2 vecteurs du réseau est encore dans le réseau ;

Slide 16

Tout réseau (entier) possède une base, i.e. une famille libre $b = (b_1, \ldots, b_r)$, tel que le réseau L est l'ensemble des combinaisons linéaires entières de vecteurs de b. Réciproquement, soit une base de r vecteurs de $\mathbb{Z}^n$ $b = (b_1, \ldots, b_r)$ avec $r \leq n$. L'ensemble des C.L. entières de ces r vecteurs est un S.G. discret de $\mathbb{Z}^n$, donc un réseau noté $L(b)$:

$$L(b) = \left\{ \sum_{i=1}^{r} b_i x_i : x_1, \ldots, x_r \in \mathbb{Z} \right\}$$

Il possède même une infinité de bases.

On dit que b est Lovász-réduite si ses vecteurs sont relativement courts :

Définition 4 *Soit $L \subseteq \mathbb{R}^n$ un réseau de base réduite $(b_1, \ldots, b_n)$*

1. *pour tout vecteur w non nul de L, $\|b_1\| \leq 2^{(n-1)/2}\|w\|$*
2. *plus généralement, pour tout ensemble $(a_1, \ldots, a_t)$ de vecteurs linéairement indépendants de L,*

$$\|b_j\| \leq 2^{(n-1)/2} \max\{\|a_1\|, \ldots, \|a_t\|\} \text{ pour } 1 \leq j \leq t$$

Slide 17

L'algorithme LLL de réduction de réseaux est un algorithme polynomial pour trouver une base réduite, étant donnée une base d'un réseau en entrée.

LLL termine après un nombre fini d'itérations :

Théorème 2 *Soit $L \subseteq \mathbb{Z}^n$ un réseau de base $(b_1, \ldots, b_n)$ et soit $C \geq 2 \in \mathbb{R}$ t.q. $\|b_i\|^2 \leq C$ pour $1 \leq i \leq n$. Le nombre d'opérations arithmétiques nécessaires à LLL est $O(n^4 \log C)$ pour des entiers de taille $O(n \log C)$ bits.*

Utilité

LLL va nous permettre de cryptanalyser le chiffre de Merke-Hellman en résolvant un problème de sous-ensemble de faible densité [9].

On définit la **densité** d'un n-uplet $(a_1, \ldots, a_n)$ comme

$$d = \frac{n}{\max\{\log a_i : 1 \leq i \leq n\}}$$

Slide 18

LLL nous permet de ramener le problème de la somme de sous-ensembles à celui de la recherche d'un vecteur court dans un réseau. LLL construit une base réduite qui contient un vecteur dont la norme est à un facteur $2^{(n-1)/2}$ du plus court vecteur du réseau.

En pratique, LLL trouve un vecteur qui est bien meilleur et LLL peut trouver une solution au problème de la somme de sous-ensembles pourvu que ce vecteur soit plus court que la plupart des vecteurs non nuls du réseau.

Résolution de SSP

Slide 19

Instance : $A = (a_1, \ldots, a_n)$, n-uplet d'entiers et un entier s.
Résultat : $x = (x_1, \ldots, x_n)$, $x_i \in \{0, 1\}$ t.q. $\langle A, x \rangle = s$.

on construit un réseau L dont la base est constituée des lignes de :

$$V = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & -a_1 \\ 0 & 1 & 0 & \ldots & 0 & -a_2 \\ 0 & 0 & 1 & \ldots & 0 & -a_3 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & -a_n \\ 0 & 0 & 0 & \ldots & 0 & s \end{pmatrix}$$

Si $v_1, \ldots v_{n+1}$ sont les lignes de V et que x est solution de SSP, alors

$$\sum_{j=1}^{n} x_j v_j + v_{n+1} = (x_1, \ldots, x_n, 0)$$

Un des vecteur trouvé par LLL est court (au sens de Lovász). Il faut donc calculer une base réduite de V et vérifier si la base réduite contient un vecteur qui et une solution au problème SSP.

Conclusion

Slide 20

Les cryptanalyses fructueuses de Merkle Hellman ont fait passer de mode le choix de problèmes NP -complets comme candidats à devenir des fonctions à sens unique.

D'autant qu'à peu près en même temps était inventé le système RSA [10] pour lequel l'hypothèse de complexité sous-jacente est l'appartenance à NP $\cap\, co - $ NP .

Cryptanalyse de Merke-Hellman (Odlyzko-LLL)

B. Martin

31/1/2002

```
> restart;
```

on construit le 5-uplet de base

```
> a:=[12,14,27,55,120];
```

$$a := [12, 14, 27, 55, 120]$$

```
> s:=0;
```

$$s := 0$$

```
> for i from 1 to 5 do s:=s+a[i] od;
```

$$s := 12$$
$$s := 26$$
$$s := 53$$
$$s := 108$$
$$s := 228$$

```
>
```

on calcule la somme des éléments et on choisit le module plus grand que la somme (228)

```
> m:=250;
```

$$m := 250$$

t est le multiplicateur

```
> t:=61;
```

$$t := 61$$

on calcule l'inverse de t par euclide étendu

```
> igcdex(t,m,'u','v');print(u,v);
```

$$1$$
$$41, -10$$

u est l'inverse de t, fixé à 31.

```
> b:=map(x->t*x mod m,a);
```

$$b := [232, 104, 147, 105, 70]$$

```
>
> readlib(lattice);
```

proc() ... **end**

on construit un sac à dos arbitraire :

```
> lattice([[1,0,0,0,0,-b[1]],[0,1,0,0,0,-b[2]],[0,0,1,0,0,-b[3]],[
  0,0,0,1,0,-b[4]],[0,0,0,0,1,-b[5]],[0,0,0,0,0,+b[1]+b[3]]],'inte
  ger');
```

$$[[1, 0, 1, 0, 0, 0], [0, -1, 0, 1, 0, -1], [-2, -1, 2, -1, 0, 0], [0, 1, 0, 1, -3, 1], [-1, 0, 2, 1, 3, 2],$$
$$[0, 3, 0, 0, 1, -3]]$$

le premier vecteur ligne est solution.

```
> lattice([[1,0,0,0,0,-b[1]],[0,1,0,0,0,-b[2]],[0,0,1,0,0,-b[3]],[
  0,0,0,1,0,-b[4]],[0,0,0,0,1,-b[5]],[0,0,0,0,0,+b[1]+b[2]]],'inte
  ger');
```

$$[[1, 1, 0, 0, 0, 0], [0, -1, 0, 1, 0, -1], [0, 0, 3, -1, 0, 0], [-1, 0, 0, 1, -3, 1], [-1, 0, 1, 2, 3, 1],$$
$$[-2, 2, -1, 0, 1, -3]]$$

```
>
```

Problèmes co-NP

Le problème obtenu par la négation de la question d'un problème de décision Π est appelé le **complémentaire** de Π.

Slide 21

La classe des problèmes de décision qui sont complémentaires d'une classe $\mathcal{C}$ de problèmes de décision sont dénommés co-$\mathcal{C}$.

Par exemple, la classe complémentaire de NP est la classe co-NP .

Les définitions de la classe co-NP peuvent se faire de la même façon que pour la classe NP . En particulier, on utilise la même notion de réduction et on peut définir comme précédemment des problèmes $co-$ NP -complets.

Le problème de la primalité

INSTANCE : Un entier n

QUESTION : n est-il premier ?

Slide 22

Il s'agit d'un problème algorithmique par excellence. Il a beaucoup été étudié et est encore loin d'être résolu.

Il existe un grand nombre de critères permettant de vérifier si un entier est premier.

Remarque : Le crible d'Eratosthene en $O(\sqrt{n})$ n'est pas polynomial. En effet, sa complexité en temps n'est pas polynomiale en la longueur de l'entrée, qui est $\log\lceil n\rceil$. Il est appelé **pseudo polynomial**.

Le test de Lucas

Théorème 3 *n est premier ssi $\exists g \in$ tel que $g^{n-1} \equiv 1(\mod n)$ et $g^{\frac{n-1}{p}} \not\equiv 1(\mod n)$ pour tout p facteur premier de $n-1$.*

Preuve [11] pour n premier, un élément primitif g modulo n remplit la condition. Réciproquement, si un tel g existe, alors l'ordre de g dans $\mathbb{Z}_n^\star$ est forcément $n-1$. Or, tout élément de $\mathbb{Z}_n^\star$ est d'ordre un diviseur de $\varphi(n)$; comme $\varphi(n) \leq n-1$, on en déduit $\varphi(n) = n-1$, ce qui est équivalent à dire que n est premier.

Slide 23

On en déduit un test de primalité appelé le test de Lucas.

Aucun des critères connus ne permettent de construire un algorithme déterministe en temps polynomial pour tester la primalité. L'algorithme le plus rapide à ce jour qui teste la primalité d'un entier n a pour complexité $O((\log n)^{c \log\log\log n})$. Il n'est pas clair qu'il n'existe pas d'algorithme déterministe en temps polynomial pour tester la primalité mais on n'en connait pas !

A partir du test de Lucas, on peut construire un algorithme non-déterministe en temps polynomial qui nous permet de dire que Primalité est dans NP .

Algorithme non-déterministe de test de primalité

Slide 24

si $n = 2$ **alors** accepte ;
si $n = 1$ **ou** $n > 2$ est pair **alors** rejette ;
si $n > 2$ est impair **alors** choisir x, $1 < x < n$;
  vérifier si $x^{n-1} \equiv 1(\mod n)$;
  deviner une factorisation de $n-1$, $\prod_1^k p_i$;
  vérifier si $n-1 = \prod_1^k p_i$;
  pour $1 \leq i \leq k$ **faire**
    vérifier si p_i est premier et $x^{\frac{n-1}{p_i}} \not\equiv 1(\mod n)$;
  fpour
  accepte **si** aucun test n'est faux.

Et le complémentaire de Primalité ?

La reconnaissance des nombres composés est dans NP .

Il suffit de construire un algorithme de type deviner/vérifier pour leur factorisation.

Plus précisément, on considère le problème FACTM des facteurs majorés.
INSTANCE : Un entier n et un nombre $M \leq n$.

Slide 25

QUESTION : Existe-t-il un diviseur de n inférieur à M ?

La difficulté de factoriser un nombre équivaut à la difficulté de résoudre FACTM. En effet, si FACTM$\in$ P, on sait factoriser n en temps polynomial par dichotomie. On cherche s'il y a un facteur de n dans l'intervalle $[1, \sqrt{n}]$ en résolvant FACTM avec $M = \lfloor \sqrt{n} \rfloor$; s'il n'y en a pas, n est premier. S'il y en a, on cherche si n a un facteur dans $[1, \sqrt{n}/2]$, sinon on cherche s'il a un facteur dans l'intervalle $[\sqrt{n}/2, \sqrt{n}]$ etc.

En conséquence,

Théorème 4 (Pratt) PRIMALITÉ$\in$ NP $\cap$ co-NP .

Quelques rappels

Indicatrice d'Euler

On appelle indicatrice d'Euler d'un entier n et on note $\varphi(n)$ le nombre d'entiers compris entre 1 et n qui sont premiers avec n [7].

$$\varphi(n) = \text{card}\{j \in \{1, \ldots, n\} : \gcd(j, n) = 1\}$$

Slide 26

On a $\varphi(1) = 1$ et pour un entier premier p, $\varphi(p) = p - 1$

Comment calculer $\varphi(n)$?

On décompose n en produit de facteurs premiers $n = \prod_{p|n, p \text{ premier}} p^{\alpha_p}$ alors,

$$\varphi(n) = \prod_{p|n, p \text{ premier}} (p^{\alpha_p} - p^{\alpha_p - 1}) = n \prod_{p|n} (1 - \frac{1}{p})$$

Exemple : $\varphi(12) = (4-2)(3-1) = 12(1-\frac{1}{2})(1-\frac{1}{3}) = 4$

Théorème de Fermat-Euler

Slide 27

On utilise le théorème d'Euler :

$$m^{\varphi(n)} \equiv 1 \mod n \text{ si } \gcd(m,n) = 1$$

qui généralise le petit théorème de Fermat :

pour p premier et m tel que $0 < m < p$,

$$m^{p-1} \equiv 1 \mod p$$

Présentation de RSA

Slide 28

1. On choisit deux entiers distincts, premiers et assez grands p et q -de l'ordre de 10^{100}
2. on fixe $n = pq$ et on publie n
3. on calcule $\varphi(n) = (p-1)(q-1)$
4. on choisit e tel que $\gcd(e, \varphi(n)) = 1$ et on le publie (clé publique, e comme *encipher*)
5. on calcule d tel que $d.e \equiv 1 \mod \varphi(n)$ (clé privée, d comme *decipher*)

Fonction de chiffrement à sens unique $E : M \mapsto M^e \mod n$.

Fonction de déchiffrement $D : C \mapsto C^d \mod m$ où d est la trappe.

Implémentations : Soit logicielles, soit matérielles (voire même mixtes).

Sur des cartes dédiées, RSA est environ 1000 fois plus lent que DES.

Sûreté de RSA et complexité

Comme exprimé dans [2], il est pratiquement impossible de faire la preuve de la sûreté de RSA.

En effet, si FACTM$\notin$ P, cela signifierait que P $\subsetneq$ NP $\cap co-$ NP .

Slide 29

Si FACTM est NP -difficile, en ce cas NP serait clos par complémentation. En effet, par l'absurde supposons FACTM NP -complet, dans $co-$ NP et NP non clos par complémentation ; alors par définition de la NP -complétude on aurait que FACTM serait simultanément dans NP et dans $co-$ NP , contredisant ainsi l'hypothèse que NP n'est pas clos par complémentation.

Ces deux conjectures sont supposés soit fausses soit très difficiles à établir (elles sont reliées à la question P= NP). Il est supposé de même pour la sûreté de RSA.

Références

[1] A. Beutelspacher. *Cryptology*. The American Mathematical Society, 1994.

[2] G. Brassard. A note on complexity and cryptography. *IEEE Trans. on Inform. Theory*, IT-25 :232–233, 1979.

[3] A.M. Cohen, H. Cuypers, and H. Sterk. *Some tapas of computer algebra*. Springer, 1999.

[4] S. Cook. The P versus NP problem. Clay Mathematics Institute.

[5] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Trans. on Inform. Theory*, 22(6) :644–654, 1976.

[6] M.R. Garey and D.S. Johnson. *Computers and intractability*. Freeman, 1979.

[7] N. Koblitz. *A course in number theory and cryptography*. Graduate texts in mathematics. Springer Verlag, 1987.

[8] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, IT-24 :525–530, 1978.

[9] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In C. Pomerance, editor, *Cryptology and Computational Number Theory*, volume 42 of *Proc. Symp. Appl. Math.*, pages 75–88, 1990.

Slide 30

[10] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Comm. of the ACM*, 21(2) :120–126, 1978.

[11] G. Zémor. *Cours de cryptographie*. Cassini, 2000.