Cours de Cryptographie

Auteur : Gérald Gavin

UCBL - Polytech - 5A informatique

Ce support du cours de cryptographie est detiné aux étudiants de 5A informatique de Polytech-lyon. L'objectif de ce cours (20h eqTD) n'est pas de formé des cryptologues mais de sensibiliser les étudiants à cette discipline et éventuellement de susciter des vocations. Les pré-requis nécessaires à la compréhension de ce fascicule sont minimes : un peu d'arithmétique et de théorie de la complexité. La rigueur est parfois volontairement sacrifiée à l'intuition et aux idées. Mais, l'enseignant, la BU ainsi qu'internet sont à votre disposition pour tout approfondissement. Il est entendu, que ce fascicule n'est qu'un support de cours et ne dispense en aucun cas les étudiants de leur présence, entre autres, pour avoir une correction des exercices et des schémas explicatifs.

Chapitre 1

Rappels d'arithmétique

Nous rappelerons les rudiments d'arithmétique modulaire utilisés dans la plupart des cryptosystèmes basés sur la factorisation. Le théorème de Fermat-Euler étant le résultat principal présenté dans ce chapitre.

Pré-requis. Résultats et notion de base en arithmétique : nombres premiers, décomposition en nombres premiers, pgcd, algorithme Euclide, ...

1.1 Rappels

- Un entier $n \in \mathbb{N} \setminus \{0, 1\}$ est premier s'il n'est divisible que par 1 et lui-même. On remarquera que 1 n'est pas premier
- Il existe une infinité de nombres premiers.
- Tout entier n s'écrit de façon unique comme produit de nombres premiers (sans tenir compte de l'ordre des facteurs).
- Le nombre d'entier premiers inférieurs à n est noté $li(n)$. On montre que :

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

Autrement dit, si l'on choisit aléatoirement un entier plus petit que n, la probabilité qu'il soit premier est proche de $1/\ln n$.

1.2 Fonction d'Euler

1.2.1 Définition

La fonction d'Euler $\phi : \mathbb{N} \to \mathbb{N}$ joue un rôle fondemental en arithmétique. $\phi(n)$ est simplement le nombre d'entiers qui sont à la fois plus petit que n et premiers avec n.

Définition 1. *La fonction d'Euler $\phi : \mathbb{N} \to \mathbb{N}$ est définie par :*

$$\phi(n) = |\{x \in \{1, \ldots, n-1\} | pgcd(x, n) = 1\}|$$

Exemple.

— $\phi(7) = |\{1, 2, 3, 4, 5, 6\}| = 6$
— $\phi(15) = |\{1, 2, 4, 7, 8, 11, 13, 14\}| = 8$

1.2.2 Comment calculer $\phi(n)$?

Le calcul de $\phi(n)$ peut se faire à l'aide de l'algorithme naif suivant :

CalculNaifPhi(n)

— $\phi := 0$
— ***pour*** $i := 1$ *à* $n - 1$
  si $pgcd(i, n) = 1$
    $\phi + +$
— ***retourner*** ϕ

Exercice 1. *Quelle est la complexité de cet algorithme (mesurée par rapport à la taille de l'entrée, i.e. le nombre de bits de la représentation binaire de n ?*

Cet algorithme est trop lent pour être utile en pratique. On peut difficilement l'utiliser pour des nombres supérieurs a 10^{15} ce qui est très en deça de nos besoins cryptographiques. Euler nous propose une formule qui nous donne $\phi(n)$ à partir de la factorisation de n.

Proposition 1. *Soit $p_1^{e_1} \cdots p_r^{e_r}$ la décomposition en nombre premiers de n. On a*

$$\phi(n) = \prod_{i=1}^{r} (p_i - 1) p_i^{e_i - 1}$$

Exercice 2. *Soit p, q deux nombres premiers. Montrer en utilisant la formule précédente que $\phi(p) = p - 1$, $\phi(pq) = (p - 1)(q - 1)$? Vérifier ceci expérimentalement.*

Si l'on choisit un entier n aléatoirement, la formule d'Euler ne nous aide pas beaucoup pour calculer $\phi(n)$ car il est nécessaire de connaitre la factorisation de n. Or il n'existe pas (à notre connaissance) d'algorithme rapide de factorisation. Cependant, cette formule nous permet de générer rapidement des couples $(n, \phi(n))$ où n est un grand entier.

Exercice 3. *Comment générer des couples $(n, \phi(n))$ avec $n = pq$ produit de deux grands entiers premiers ?*

1.3 Théorème de Bezout

Le théorème de Bézout est un resultat fondamental en arithmétique. La plupart des résultats simples peuvent être démontrés en utilisant ce résulat.

Théorème 1. *Soient $a, b \in \mathbb{Z}$. On a $pgcd(a, b) = 1$ si et seulement si $\exists u, v \in \mathbb{Z}$, tels que $au + bv = 1$.*

Démonstration. ($\Rightarrow$) Supposons $pgcd(a,b) = 1$. Notons D l'ensemble des entiers de la forme $au + bv$. Si n et m appartiennent à D, alors le reste de la division de n par m (qui s'écrit $n - qm$) aussi. Donc tous les reste r_i obtenus avec l'algo d'Euclide (revoir cet algo) appartiennent à D donc $pgcd(a,b) = 1$ appartient à D.

($\Leftarrow$) Supposons $\exists u, v \in \mathbb{Z}$ tels que $au + bv = 1$. Raisonnons par l'absurde en supposant que $pgcd(a,b) = d > 1$. Comme a et b sont multiples de d, tout élement de D est aussi multiple de d. Donc 1 n'est pas élément de D. Contradiction.

□ □

Exercice 4. *Trouver $u, v \in \mathbb{Z}$ dans les cas $a = 10, b = 21$ et $a = 10, b = 15$.*

Exercice 5. *Montrer que si $pgcd(a,b) = d$ alors $\exists u, v \in \mathbb{Z}$, tels que $au + bv = d$.*

Exercice 6. *Montrer que si $pgcd(a,b) = d$ alors $\nexists u, v \in \mathbb{Z}$, tels que $au + bv = d' < d$.*

Les entiers u, v sont appelés coefficients de Bézout. Il reste à savoir comment les trouver efficacement pour des grands nombres. Le fameux algorithme d'Euclide permet de trouver efficacement le *pcgd* de deux entiers. Il peut être *étendu* pour renvoyer aussi les coefficients de Bézout.

EuclideEtendu(a,b)

- **si** $b = 0$
  retourner $(a, 1, 0)$
- $(d', x', y') \leftarrow EuclideEtendu(b, a \mod b)$
- $(d, x, y) \leftarrow (d', y', x' - \lfloor a/b \rfloor y')$
- **retourner** (d, x, y)

Exercice 7. *Vérifier cet algo "à la main".*

Le théorème de Bézout est notamment à la base du théorème de Gauss.

Théorème 2. ***(Gauss).*** *Soient a, b, c des entiers non-nuls tels que $pcgd(a,b) = 1$. Si $a|bc$ alors $a|c$*

Démonstration. D'après Bezout, $\exists u, v \in \mathbb{Z}$ tq $au + bv = 1$.
Donc $auc + bvc = c$
Or $a|bc \Rightarrow \exists k \in \mathbb{Z}$ tel que $bc = ka$
Donc $a(uc + vk) = c$
$\Rightarrow a|c$
□ □

Corollaire. Si $a|bc$ et a premier alors $a|b$ ou $a|c$.

1.4 Congruence

Dans cette section, nous généralisons que ce nous connaissons sur les nombres pairs et impairs (dans ce cas on travaille *modulo* 2). Par exemple, la somme de deux nombres pairs ou impairs est paire. Ceci s'ecrira $0 + 0 \equiv 1 + 1 \mod 2$. De même, le produit de deux nombres pairs est pair, celui de deux nombres impairs est impairs et celui d'un nombre pair et impair est pair. Ceci s'écrit respectivement par $0 \times 0 \equiv 0 \mod 2$, $1 \times 1 \equiv 1 \mod 2$ et $0 \times 1 \equiv 0 \mod 2$.

1.4.1 Définition

Soient a, b, n trois entiers (on pourra supposer n strictement positif). La notation $a \equiv b \pmod n$ (qui se lit "*a congru à b modulo n*") signifie que $a - b$ est un multiple de n ou de manière équivalente

$$a \equiv b \pmod n \Leftrightarrow \exists k \in \mathbb{Z}, a = b + kn$$

On verifie que cette relation de congruence est une relation d'équivalence :

1. $a \equiv a \pmod n$ (réflexivité)
2. $a \equiv b \pmod n \Rightarrow b \equiv a \pmod n$ (symétrie)
3. $a \equiv b \pmod n$ et $b \equiv c \pmod n \Rightarrow a \equiv c \pmod n$ (transitivité)

Exercice 8. *Vérifier que cette relation (modulo n) d'équivalence possède n classes d'équivalence et que les entiers* $0, 1, \ldots, n-1$ *appartiennent à des classes d'équivalence différentes.*

De plus, cette relation d'équivalence est compatible avec les operations arithmétiques :

Proposition 2. *Si* $a \equiv a' \pmod n$ *et si* $b \equiv b' \pmod n$ *alors :*
— $a + b \equiv a' + b' \pmod n$
— $ab \equiv a'b' \pmod n$

Exercice 9. *Prouver ce résultat et testez-le sur quelques exemples.*

Ce résultat signifie (en vulgarisant quelque peu) que l'évaluation d'une expression arithmétique *modulo n*, il suffit de connaitre les variables *modulo n*.

1.4.2 Lien entre "*modulo*" et "*reste de la division*"

Une fois n'est pas coutume, nous allons introduire de l'ambiguité en notant $a \mod n$ le reste de la division de a par n. Autrement dit, $a \mod n$ est l'unique entier de $\{0, \ldots, n-1\}$ vérifiant $a = kb + (a \mod n)$ avec $k \in \mathbb{Z}$. La proposition suivante indique le lien entre les deux emplois de mod .

Proposition 3. *On montre que :*

1. $a \mod n \equiv a \pmod n$
2. $a \equiv b \pmod n$ *si et seulement si* $a \mod n = b \mod n$

Exercice 10. *Prouver ce résultat.*

En couplant ceci avec le proposition 2, on peut écrire que :

Proposition 4. *On montre que :*

1. $(a+b) \mod n = (a \mod n + b \mod n) \mod n$
2. $(ab) \mod n = ((a \mod n)(b \mod n)) \mod n$

Exercice 11. *Prouver le resultat précédent. On pourra utiliser le résultat évident suivant : si $a \equiv b \pmod n$ et $0 \leq a, b < n$ alors $a = b$.*

En vulgarisant légèrement, lorsque l'on veut calculer une expression arithmétique modulo n, on peut calculer normalement (sur $\mathbb{Z}$) et faire un modulo n à la fin (reste de la division par n) ou après chaque opération. Ceci peut permettre d'accélerer les calculs en empêchant que les nombres deviennent trop grands.

Exemple 1. $(2059*9876*1234+1004) \mod 5 = (4*1*4+4) \mod 5 = (16 \mod 5+4 \mod 5) \mod 5 = (4+1) \mod 5 = 0$

Exercice 12. *Calculer "à la main", en utilisant ce qui précède, $4^{2019} \mod 5$.*

1.5 Théorème de Fermat-Euler

1.5.1 The résultat

Commençons par une version restreinte du théorème de Fermat.

Théorème 3. ***(petit théorème de Fermat).*** *Pour tout $x \in \mathbb{Z}$ et tout entier n premier*

$$x^n \equiv x \pmod n$$

Démonstration. Supposons $x \geq 0$ et raisonnons par récurrence. Si $x = 0$, la proposition est vraie. Supposons que $x^n \equiv x \mod n$ (hypothèse de récurrence). Comme $(x+1)^n = x^n + n(\ldots) + 1$ (binôme de Newton), on a bien $(x+1)^n \equiv x^n + 1 \equiv x + 1 \pmod n$.
□ □

Exemple 2. $4^5 = 1024 \equiv 4 \pmod 5$.

Le corollaire du résultat précédent est que

$$x^{n-1} \equiv 1 \pmod n$$

Ce résultat peut-il se généraliser à des n non premiers ? La réponse est oui mais en interprétant $n-1$ comme égal à $\phi(n)$ lorsque n est premier.

Théorème 4. *(**Euler-Fermat**). Pour tout entier x premier avec n,*

$$x^{\phi(n)} \equiv 1 \pmod n$$

Démonstration. (pour $n = pq$). Dans ce cas, $\phi(pq) = (p-1)(q-1)$. Comme x est premier avec n, x est premier avec p et q. D'après le corollaire précédent, $x^{p-1} \equiv 1 \pmod{p}$ et $x^{q-1} \equiv 1 \pmod{q}$. Donc $x^{\phi(n)} \equiv 1 \pmod{p}$ et $x^{\phi(n)} \equiv 1 \pmod{q}$. Autrement dit, $x^{\phi(n)} - 1$ est un multiple de p et de q, donc un multiple de $n = pq$. Ceci montre le résultat.

□

Exercice 13. *Montrer que :*

$$x^e \equiv (x \mod n)^{e \mod \phi(n)} \pmod{n}$$

Ce résultat est fondamental en arithmétique modulaire et à la base de la plupart des cryptosystèmes basés sur le problème de la factorisation (c'est à dire presque toute la cryptographie sur internet). Nous verrons ceci dans les chapitres suivants. Commencons par voir une application algorithmique.

1.5.2 Test de Fermat

Le problème de tester si un entier est premier date de l'antiquité. Jusqu'au moyen-âge (avant Fermat), pour décider si un entier n est premier, on cherchait un facteur $1 < p < n$. Clairement n était décidé premier si l'on ne trouvait pas de tels facteurs. Autrement dit, le problème de la primalité et celui de la factorisation se traitaient avec les mêmes algorithmes (non efficaces pour des grands nombres). Les résultats précédents ont changé les choses. Le problème de la factorisation reste un problème difficile (il n'existe toujours pas d'algorithme efficace pour factoriser des grands nombres) mais celui de la primalité devient facile avec le test suivant :

TestFermat(n)
  retourner ($2^{n-1} \mod n == 1$)

Ce test (très simple) est efficace car $2^{n-1} \mod n$ peut se calculer rapidement (même pour des nombres de 1000 chiffres), voir chapitre 2. Cependant, ce test commet des erreurs. En effet des nombres composites (non premiers) sont décidés premiers.

Exercice 14. *Trouver le plus petit entier $n \geq 2$ mal prédit.*

Exercice 15. *Proposez un algorithme efficace pour générer des nombres premiers de taille k.*

Ce test sera néanmoins suffisant pour nos besoins. En effet, la probabilité de se tromper diminue avec la taille des nombres testés. Par exemple, pour des nombres de 300 chiffres, la probabilité d'erreurs est inférieure à 10^{-60} et peut être négligée. Il est à noter que tous les tests de primalité efficaces existants sont basés sur le théorème de Fermat et que le premier test déterministe date seulement de 2004.

1.6 Arithmétique Modulaire

L'arithmétique modulaire est juste une façon de réécrire la section sur les congruences et pourrait être appelé "arithmétique des congruences". Autrement dit, il n'y a aucune idée nouvelle dans cette section. Formellement, $\mathbb{Z}/n\mathbb{Z}$ est le quotient de l'anneau [1] $\mathbb{Z}$ par l'idéal $n\mathbb{Z}$. L'anneau $\mathbb{Z}/n\mathbb{Z}$ contient les n classes d'équivalence de la relation d'équivalence "*modulo* n", i.e. les n ensembles de la forme $\{i + kn | k \in \mathbb{Z}\}$ qui peuvent être identifiés à $\{0, \ldots, n-1\}$. On munit ensuite cette structure de l'addition $\oplus$ et la mulitplication $\otimes$ définie par :

— $a \oplus b = (a + b) \mod n$

— $a \otimes b = (a \times b) \mod n$

où $+$ et $\times$ sont l'addition et la multiplication classiques sur $\mathbb{Z}$.

Exemple 3. *Dans* $\mathbb{Z}/5\mathbb{Z}$, $3 \oplus 4 = 2$ *et* $3 \otimes 4 = 2$.

Il est à montrer que les opérations dans $\mathbb{Z}/n\mathbb{Z}$ suivent les mêmes règles que dans $\mathbb{Z}$.

Proposition 5. *Soient* $a, b, c \in \mathbb{Z}/n\mathbb{Z}$.

1. $a \oplus 0 = a$
2. $1 \otimes a = a$
3. $a \oplus b = b \oplus a$
4. $a \otimes b = b \otimes a$
5. $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
6. $(a \otimes b) \otimes c = a \otimes (b \otimes c)$
7. $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$

Démonstration. Découle de la proposition 2. Montrons, par exemple, la distributivité de la multiplication / addition, i.e la propriété 7.

$a \otimes (b \oplus c) = (a((b + c) \mod n)) \mod n$ (par définition des opérations $\oplus$ et $\otimes$)
Donc $a \otimes (b \oplus c) = a(b + c) \pmod n = (ab + ac) \mod n = ((ab \mod n) + (ac \mod n)) \mod n = (a \otimes b) \oplus (a \otimes c)$ d'après la proposition 4.

□

1.6.1 Eléments inversibles

Un élément $x \in \mathbb{Z}/n\mathbb{Z}$ est l'inverse d'un élément $y \in \mathbb{Z}/n\mathbb{Z}$ si $x \otimes y = 1$.

Proposition 6. *On a les propriètés suivantes :*

1. *Un élément* $x \in \mathbb{Z}/n\mathbb{Z}$ *a au plus un inverse. S'il en a un, son inverse est noté* x^{-1}.
2. *Un élément* $x \in \mathbb{Z}/n\mathbb{Z}$ *est inversible si et seulement si* $pgcd(x, n) = 1$.

1. Informellement, un anneau est une structure muni de l'addition et la multiplication qui respectent les lois classiques de l'arithmétique.

3. Le cardinal de l'ensemble des élements inversibles, noté $(Z/n\mathbb{Z})^*$, est égal à $\phi(n)$.

Démonstration. 1 - Pour simplifier, supposons n premier. Raisonnons par l'absurde en supposant qu'il existe $y, y' \in \mathbb{Z}/n\mathbb{Z}$ tel que $y \neq y'$ et $x \oplus y = x \oplus y' = 1$. Donc $xy \equiv xy' \equiv 1 \pmod{n}$, par définition de la multiplication dans $\mathbb{Z}/n\mathbb{Z}$. Ceci est équivalent à $xy = 1 + kn$ et $xy' = 1 + k'n$. Il s'en suit que $x(y - y') = k''n$. Or $0 \leq x < n$ et $y - y' \in \{-n+1, \ldots, n-1\} \setminus \{0\}$ ce qui implique que x , ni $y - y'$ ne possède n comme facteur . Donc $x(y - y')$ n'est un multiple de n d'après le lemme de Gauss.

2 - $x \otimes y = 1$
$\Leftrightarrow xy = 1 + kn$
$\Leftrightarrow xy - kn = 1$
$\Leftrightarrow pgcd(x, n) = 1$ d'après le théorème de Bezout ($y, -k$ sont des coefficients de Bezout).

3 - Par définition $\phi(n) = \{x \in \{0, \ldots, n-1 | pgcd(x, n) = 1\}$.

□

Exemple 4. *3 est l'inverse de 2 dans $\mathbb{Z}/5\mathbb{Z}$.*

Exemple 5. *3 n'a pas d'inverse dans $\mathbb{Z}/15\mathbb{Z}$.*

Exercice 16. *Explicitez l'ensemble $(\mathbb{Z}/15\mathbb{Z})^*$*

Exercice 17. *Montrer que $n - 1$ est toujours son propre inverse dans $\mathbb{Z}/n\mathbb{Z}$.*

Il reste à trouver une méthode efficace pour calculer l'inverse d'un élément de $(Z/n\mathbb{Z})^*$. L'algorithme d'Euclide étendu le permet.

Proposition 7. *Soient $x \in (\mathbb{Z}/n\mathbb{Z})^*$ et $(d, a, b) \leftarrow Euclide - Etendu(x, n)$.*

$$x^{-1} = a \mod n$$

Démonstration. D'après la proposition 6, si x est inversible alors $d = pgcd(x, n) = 1$. Donc, $ax + bn = 1 \Leftrightarrow ax = 1 - bn \Leftrightarrow ax \equiv 1 \pmod{n} \Leftrightarrow (a \mod n)x \equiv 1 \pmod{n} \Leftrightarrow (a \mod n)x \mod n = 1 \Leftrightarrow x^{-1} = (a \mod n)$ □

Exercice 18. *Soit $x \in (\mathbb{Z}/n\mathbb{Z})^*$. Montrer que $x^{-1} = x^{\phi(n)-1} \mod n$*

1.6.2 Arithmétique modulaire vs arithmétique

L'arithmétique modulaire est une structure finie, donc manipulable par un ordinateur. De plus, cette structure permet de faire des calculs car elle est munie d'une addition et multiplication. En calculant dans $\mathbb{Z}/n\mathbb{Z}$, on calcule modulo n. Ceci peut aussi permettre de calculer dans $\mathbb{Z}$ en utlisant la relation évidente suivante

$$x \in \{0, \ldots, n-1\} \Rightarrow x \mod n = x$$

Concrètement, imaginons que l'on souhaite évaluer une expression arithmétique $C(x_1, \ldots, x_t)$ dans $\mathbb{Z}$. Si l'on sait que $C(x_1, \ldots, x_t) \in \{0, \ldots, n-1\}$ (si par exemple $x_1, \ldots, x_t \ll n$), alors évaluer $C(x_1, \ldots, x_t)$ dans $\mathbb{Z}$ ou $\mathbb{Z}/n\mathbb{Z}$ est équivalent.

Remarque 1. *Une fois que l'on a précisé que l'on travaille dans* $\mathbb{Z}/n\mathbb{Z}$*, on n'a plus besoin d'utiliser* $\equiv$ *et* $\mod$ *qui sont remplacés par l'égalité* $=$*. Autrement dit, si* $a, b, c \in \mathbb{Z}/n\mathbb{Z}$*,* $a \oplus b = c$ *signifie que* $a + b \equiv c \mod n$*. De plus, dans la suite,* $\oplus$ *et* $\otimes$ *seront remplacés par* $+$ *and* $\times$*. Donc, en prenant soin de préciser que l'on travaille dans* $\mathbb{Z}/5\mathbb{Z}$ *(par exemple), on pourra écrire* $3 + 4 = 2$*.*

Chapitre 2

Digressions algorithmiques

Nous présenterons dans cette fiche quelques considérations algorithmiques utiles en cryptographie.

Pré-requis. Notions élémentaires de complexité

En cryptographie, la complexité des algorithmes est traitée de manière relativement binaire. Les algorithmes de complexité polynomiale seront considérés comme efficaces (rapides) et ceux de complexité non polynomiale seront considérés comme inutilisables.

2.1 Complexité polynomiale

Soit $f : \{0,1\}^* \to \{0,1\}^*$. Un algorithme A *implémente* f s'il retourne $f(e)$ sur une entrée e, i.e. $A(e) \to f(e)$. La taille d'une entrée e sera notée $|e|$. Elle correspond au nombre de bits utilisés pour écrire e. Par exemple, pour une fonction $f : \mathbb{N} \to \mathbb{N}$, la taille de l'entrée $x \in \mathbb{N}$ est le nombre de bits x (nombre de chiffres de x écrit en binaire) et non x lui-même.

Exercice 19. *Montrer que la taille d'un entier $x \in \mathbb{N}$ est $\lfloor \log_2 x \rfloor + 1$.*

Classiquement, la complexité est une notion asymptotique permettant de borner le temps d'exécution en fonction de la taille de l'entrée.

Définition 2. *La complexité d'un algorithme A est polynomiale si le temps d'execution $T_A(e)$ de $A(e)$ peut être borné par $p(|e|)$ pour un certain polynôme p (ou de manière équivalente s'il existe d tel que $T_A(e) = O(|e|^d)$).*

Remarque 2. *On peut remplacer "temps d'exécution" par "nombre d'opérations élémentaires / processeur" dans la définition précédente.*

Exercice 20. *Justifier le fait que la complexité l'algorithme classique de multiplication (appris à l'école primaire) est polynomiale.*

Exercice 21. *Soit la fonction $Exp : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ définie par $Exp(a,b) = a^b$. Evaluer la taille (en nombre de bits) de a^b en fonction de la taille de a et de b. En déduire, qu'il n'existe pas d'algorithme de complexité polynomiale implémentant Exp.*

2.2 Complexité polynomiale vs cryptographie

Contrairement aux apparences, la cryptographie est une discipline purement informatique (et non mathématique). En effet, dans la grande majorité des cas, un schéma cryptographique est dit sûr s'il n'existe d'attaque de complexité polynomiale. Autrement dit, il existe presque toujours un algorithme capable de *casser* la sécurité mais s'il est trop long a exécuter et/ou s'il réussit avec une probabilité trop petite alors on n'en tiendra pas compte.

Plus formellement, un schéma cryptographique devra s'exécuter en temps polynomial et sera dit sûr il n'existe pas d'attaque de complexité polynomiale ou pour être précis toute attaque de complexité polynomiale ne réussit qu'avec une probabilité négligeable, ie. $O(2^{-\lambda})$

Paramètre de sécurité λ. En cryptographie, on considère traditionnellement un paramètre de securité $\lambda \in \mathbb{N}$. Ce paramètre est généralement fixé par des instances (indépendantes) composées de cryptologues et plus généralement de spécialistes en informatique. Le choix de ce paramètre dépend des ressources informatiques suceptibles d'être mobilisées par les attaquants aujourd'hui et dans les années à venir. Concretement, ce paramètre est choisi de telle sorte qu'il est garanti qu'un attaquant ne puisse pas effectuer 2^λ opérations élémentaires / processeur.

Exercice 22. *Justifier le fait qu'actuellement, on choisit généralement* $\lambda = 100$.

Exercice 23. *Que dire d'une attaque qui nécessite* $t(\lambda)$ *opérations élémentaires et qui ne réussit qu'avec une probabilité de* $p(\lambda)$ *avec* $t(\lambda)p(\lambda) \geq 2^\lambda$ *?*

Evidemment, ce choix doit prendre en compte les évolutions technologiques prochaines et deviendrait immédiatemment caduque si, par exemple, des ordinateurs quantiques voyaient le jour.

Concrétement, lorsque l'on développe un protocole cryptographique, il doit être paramétré (taille des clés par exemple) de manière à ce que toute attaque requiert au moins 2^λ opérations élémentaires.

2.3 Algorithme de Factorisation

Le problème de la factorisation est un problème important en cryptographie. En effet, presque toute la sécurité sur internet repose sur ce problème. Un algorithme de factorisation prend un entier n en entrée et retourne un facteur non-trivial de n, si n n'est pas premier. L'algorithme naif de factorisation consiste à tester tous les facteurs possibles.

AlgoNaifFactorisation(n)

- $i \leftarrow 2$
- **tant que** $i < n$
  si $n \mod i == 0$
  retourne i
  $i++$

— **retourne** $\perp$

Clairement, le nombre d'itérations, dans le pire des cas, est en $\Omega(n)$. Peut-on en conclure que la complexité de cet algorithme est polynomiale ? La réponse est évidemment non. En effet, il faut mesurer le temps d'exécution par rapport à la taille de l'entrée. Or la taille de l'entrée $t_n = \lfloor \log_2 n \rfloor + 1 \geq \log_2 n$, ce qui signifie que le temps d'exécution est en $\Omega(2^{t_n})$ (on rappelle que $2^{\log_2 n} = n$). On a ainsi montrer que la complexité de *AlgoNaifFactorisation* est exponentielle et non polynomiale. Pour s'en rendre compte, il suffit de constater qu'il ne permet pas de traiter des nombres ayant des centaines de chiffres.

Exercice 24. *Evaluer le temps d'exécution de AlgoNaifFactorisation(n) lorsque n est un entier premier de 100 chiffres.*

Exercice 25. *Proposer un amélioration de AlgoNaifFactorisation. Obtenez-vous une complexité polynomiale ?*

A notre connaissance, il n'existe pas d'algorithme de factorisation efficace. Mais nous ne pouvons pas affirmer qu'il n'en existe pas. Même $P \neq NP$ ne garantirait rien car le problème de la factorisation n'est pas NP-difficile. La seule justification de la difficulté de ce problème est que l'on n'a pas trouvé d'algorithme efficace en 3000 ans de recherche.

Conjecture 1. *Soit $n = pq$ un produit de deux nombres premiers p et q de taille k choisis aléatoirement. Il n'existe pas d'algorithme polynomial A tel que $A(n)$ retourne un facteur de n avec une probabilité non-négligeable.*

Cette conjecture est à la base de 95% de la sécurité sur internet.

2.4 Exponentiation modulaire

L'exponentiation modulaire consiste à calculer $a^b \mod c$ étant donnés a, b, c. La question est de savoir s'il existe un algorithme (de complexité) polyomial implémentant cette fonction. On a vu dans l'exercice 21 que la fonction a^b ne peut pas s'implémenter en temps polynomial car les nombres deviennent trop grand. Ce problème n'a plus court pour l'exponentiation modulaire car $a^b \mod c \in \{0, \ldots, c-1\}$. On pourrait donc proposer l'algorithme suivant :

ExpModNaif(a,b,c)

— $r \leftarrow 1$
— **pour** $i = 1$ à b
  $r \leftarrow (r * a) \mod c$
— **retourne** r

Cet algorithme est correct mais sa complexité n'est pas polynomiale. En effet, comme il y a b iterations, sa complexité est en $O(b) = O(2^{\log_2 b})$, ce qui est exponentielle par

rapport à la taille de l'entrée. Il faut donc réduire de nombre d'irérations. L'idée est très simple. Par exemple, si vous souhaitez calculez a^{16}, vous pouvez faire $a \times a \times a \ldots$, ce qui nécessite 15 multiplications. Vous pouvez réduire ce nombre de multiplications à 4 en utilisant l'égalité $a^{16} = a^{2^{2^{2^2}}}$. C'est cette idée qui exploitée dans l'algorithme suivant :

$ExpMod(a, b = (b_n. \ldots .b_0)_2, c)$

$e = 1$
pour $k \leftarrow n$ à 0
  $e = e^2 \mod c$
  if $b_k = 1$
    $e \leftarrow (e \times a) \mod c$
retourner e

Le nombre d'itérations est maintenant en $\log_2 b$. Comme toutes les opérations sont en $O(\log_2^2 c)$, la complexité de cet algorithme est en $O(\log_2 b \times \log_2^2 c)$, qui est donc polynomiale. Il reste à prouver que cet algorithme est correct.

Proposition 8. *ExpMod est correct*

Démonstration. Montrons par récurrence sur le nombre d'itération que *ExpMod* est correct (calcule bien $a^b \mod c$). On appelle P_i la propriété suivante : « $e = a^{(bn,\ldots,bn-i+1)2} \mod c$ à la fin de la ième itération ». Il s'agit donc de montrer que P_{n+1} est vraie.

- P_1 est vraie. En effet si $b_n = 0$ alors $e = 1$ et sinon $e = a \mod c$
- $P_i \Rightarrow P_{i+1}$
  P_i implique que $e = a^{b'} \mod c$ avec $b' = (bn, \ldots, bn - i + 1)_2$. Donc à la fin de la $(i+1)$ème itération, $e = (a^{b'} \mod c)^2 \times a^{b_{n-i}} \mod c = a^{2b'+b_{n-i}} \mod c$ On obtient le résultat en remarquant que $(b_n, \ldots, b_{n-i})2 = 2b' + b_{n-i}$.

□

Ceci montre que le test de Fermat, vu précédemment, peut être implémenté en temps polynomial. On peut donc tester des très grands nombres rapidement. Ceci servira dans tous les cryptosystèmes basés sur la factorisation qui nécessitent la génération de grands nombres premiers. De plus, dans la plupart d'entre eux, l'exponentiation modulaire servira dans l'encryption et/ou la decryption.

2.5 Reduction polynomiale

La réduction en temps polynomial d'un algorithme à un autre algorithme est une activité fondamentale en algorithmie. Elle consiste à comparer la difficulté d'un problème à celle d'un autre. En cryptographie, elle nous permettra d'affirmer des choses comme par exemple "casser la sécurité de ce protocole" est aussi difficile que de factoriser des grands nombres. Or, comme il communément admis que factoriser des grands nombres est difficile, on peut croire en la sécurité du protocole. Il n'y aura jamais de sécurité absolue mais seulement une sécurité relative issue de réductions.

Proposons une définition informelle de la notion de réduction. On souhaite montrer qu'un problème P_B est aussi difficile qu'un probleme P_A dans le pire des cas, on notera ceci $P_A \leq P_B$. Pour cela, on suppose qu'il existe un algorithme rapide (de complexité polynomiale) pour résoudre le problème B. Ensuite, on essaie de construire un algorithme A, pouvant utiliser B, pour résoudre le problème P_A en temps polynomial. Un tel algorithme A s'appelle *une réduction*. L'existence d'une réduction permet d'affirmer : "*si l'on sait résoudre P_B efficacement alors on sait aussi résoudre P_A efficacement (en temps polynomial)*". C'est la contraposée de cette affirmation qui nous intéresse, i.e. "**si l'on ne sait pas résoudre P_A efficacement alors on ne sait pas non plus résoudre P_B éfficacement**". Si P_A un problème prouvé difficile (NP-complet par exemple) alors on prouve que P_B est aussi difficile. Si P_A est un problème que tout le monde pense difficile (e.g. la factorisation) alors tout le monde pense (ou devrait penser) aussi que P_B est difficile.

Exemple de réduction. Soit $n = pq$ un produit de deux nombres premiers. On souhaite montrer que trouver $\phi(n)$ à partir de n (P_B) est aussi difficile que de factoriser n (P_A). On suppose donc qu'il existe un algorithme polynomial B tel que $B(n) \to \phi(n)$. l'algorithme A suivant est une réduction du problème A au problème B.

A(n)

$\phi \leftarrow B(n)$
$S \leftarrow n - \phi + 1$
retourner $(S + \sqrt{S^2 - 4n})$

Une telle réduction permettra d'affirmer qu'un attaquant ne peut pas retrouver $\phi(n)$ à partir de n en admettant que la factorisation est un problème difficile. Autrement dit, ϕ pourra être considéré comme secret et pourra donc être utilisé pour décrypter.

Exercice 26. *Prouver que la réduction précédente est correcte, à savoir que A est polynomial et qu'il retourne un facteur de n.*

Exercice 27. *D'après l'exercice 18, l'inverse modulaire de x est égal à $x^{\phi(n)-1} \mod n$. Cette méthode, consistant à calculer $x^{\phi(n)-1} \mod n$, est-elle efficace pour calculer l'inverse modulaire de x ?*

Exercice 28. *Soit $n = pq$ un produit de deux grands nombres premiers choisis aléatoirement. Existe-t-il un algorithme polynomial A tel que $A(n)$ retourne un élément non-nul et non-inversible de $\mathbb{Z}/n\mathbb{Z}$?*

2.6 Informatique quantique

Les récents progrès technologiques laissent à penser que des ordinateurs quantiques pourraient prochainement voir le jour. Ceci mettrait à mal toute la cryptographie actuelle sur internet basée sur le problème de la factorisation. En effet, il a été montré que ce problème peut être résolu en temps polynomial avec un ordinateur quantique. Il faut donc inventer une cryptographie basé sur des problèmes supposés (ou prouvés) difficiles

dans ce nouveau monde. Voici quelques exemples de problèmes qui sembleraient rester difficiles :

— résoudre des systèmes d'équations non-linéaires

— trouver de petites solutions dans un système d'équations linéaires (SIS)

— résoudre des systèmes d'équations linéaires buitées (LWE)

— trouver des vecteurs courts dans des réseaux euclidiens (SVP)

Chapitre 3

Cryptographie symétrique

Nous parlerons de cryptographie symétrique et nous tirerons quelques premières conclusions.

Pré-requis. Un QI strictement positif.

3.1 cryptographie préhistorique

Depuis l'antiquité, l'homme cherche à transmettre de l'information de manière sécurisée. Nous passerons l'époque où la cryptographie consistait à couper la langue du messager ou à écrire sur son crane rasé et attendre la repousse. Commençons par le code césar.

3.1.1 Code de César.

Ce code consistait à effectuer un décalage de chaque lettre du message par une permutation circulaire convenue au préalable. Il y a donc 25 chiffrements possibles si l'on exclut le chiffrement consistant à ne rien faire. Ainsi, pour envoyer un message m à Cléopatre, César chiffrait le message $m \to m'$ et chargeait un messager de transmettre le chiffré m' à sa belle qui le déchiffrait $m' \to m$. Le chiffrement guarantit que m' ne resemble pas du tout à m et ne peut donc pas être exploité pour retrouver m.

Exemple 6. *m ='amour'$\to$ m' ='bnpvs' avec un décalage de 1.*

Exercice 29. *Proposez un contexte permettant d'utiliser cette méthode de chiffrement.*

Exercice 30. *Montrer que César ne pouvait pas utiliser cette même méthode avec une autre de ses maîtresses*

De manière générique, le décalage secret utilisé sera appelé la clé secrete. Dans le chiffrement de César, le nombre de clés possibles est trop faible. Ainsi si un attaquant (le messager ou autre) connait le chiffrement de César, il peut essayer toutes les clés et retrouver m à partir de m'. Il faudrait que César garde secret l'algoritme de chiffrement utilisé. Cependant, dans ce cas, cet algorithme ne peut être utilisé qu'avec Cléopatre ce qui en limite grandement son intéret.

Conclusion 1. *On ne peut pas baser la sécurité en comptant sur le fait que l'algorithme de chiffrement est secret*

Conclusion 2. *On appelle clé, un paramètre tenu secret utilisé par l'algorithme de chiffrement. Le nombre de clés doit être grand de manière à ce qu'elles ne puissent pas toutes être testées par force brute.*

3.1.2 Améliorations du code de César

Le code de César peut s'étendre facilement de manière à augmenter le nombre de clés. Par exemple, au lieu de décaler les lettres, on peut choisir une permutation quelconque, i.e. $a \to t, b \to e, c \to z \ldots$. En faisant ceci, le nombre de clés augmente drastiquement. En effet, il devient égale à $26\,!$ rendant inopérant tout algorithme type "*force brute*", même avec les ressources d'aujourd'hui. Un petit inconvénient étant que la taille de la clé augmente et qu'il deviendrait difficile pour Cléopatre de l'apprendre par coeur. Elle devrait donc la noter et prendre ainsi le risque.

Conclusion 3. *Il faut se prémunir du risque d'intrusion, c'est à dire être capable de conserver une information secrète, sinon il n'y pas de cryptographie possible.*

De plus, bien qu'il ne soit plus possible de tester toutes les clés, cette façon de chiffrer (ce cryptosystème) n'est pas sûr. En effet, l'analyse fréquentielle exploitant les régularités statistiques de la langue, e.g. la lettre la plus fréquente est le 'e', permet de *casser* ce type de chiffrement. '*Casser*' signifie que l'attaquant peut retrouver parfois le message. Même si l'attaquant ne peut retrouver le message que dans des circonctances très particulières et que cela ne se produit qu'avec une petite probabilité (mais pas trop petite), e.g. de 0.0001%, on considère que le chiffrement n'est pas sûr.

Conclusion 4. *Un cryptosystème est considéré comme sûr si un attaquant ne peut retrouver le message qu'avec une probabilité négligeable, e.g. inférieure à* 10^{-30}.

Exercice 31. *La probabilité de gagner 4 fois de suite au loto est-elle supérieure ou inférieure à* 10^{-30} *?*

3.2 Cryptographie symétrique moderne

Afin d'éviter les attaques par analyses fréquentielles, un message m sera au préalable transformé binaire (en utilisant le code ASCII par exemple). On pourra donc supposer que $m \in \{0,1\}^*$. Ce mot sera ensuite décomposé en blocs $m_1, m_2, \ldots$ de même taille ℓ et chaque bloc sera chiffré avant d'être envoyé. Il s'agit donc ici de trouver une methode pour chiffrer un mot binaire de taille ℓ. Nous analyserons deux idées intuitives utilisés dans le cryptosystèmes actuels, e.g. AES.

3.2.1 Masquer

L'idée la plus simple pour chiffrer $m \in \{0,1\}^\ell$ serait de changer certains bits choisis aléatoirement. Ceci revient à appliquer un masque binaire $\mu \in \{0,1\}^\ell$ choisi

aléatoirement sur le message m. Plus précisement, le chiffrement de m est le message [1] $m' = m \oplus \mu$, i.e. $m' = (m_1 \oplus \mu_1, \ldots, m_\ell \oplus \mu_\ell)$. La clé est donc le masque μ.

Exercice 32. *Quelle est le nombre de clés ? En déduire un choix pour ℓ.*

Exercice 33. *Comment déchiffre-t-on ?*

La question délicate qui se pose maintenant est de savoir si cette méthode de chiffrement est sûr (l'utiliseriez-vous ?). Pour répondre à cette question il faut donner des pouvoirs théoriques à l'attaquant surestiment ses pouvoirs réels. Si l'on montre que le chiffrement est sûr face à un tel super-attaquant alors il est aussi sûr dans le monde réel.

Conclusion 5. *Pour analyser la sécurité d'un cryptosystème, il faut définir formellement les pouvoirs de l'attaquant.*

Traditionnellement, on suppose que l'attaquant à accès à un oracle d'encryption. Autrement dit, on supposera que l'attaquant peut choisir des messages et obtenir les encryptions. Si le cryptosystème est sûr contre un tel attaquant on parlera de sécurité contre des attaques à messages choisis (*CPA secure* en anglais).

Exercice 34. *Proposer un scénario dans lequel l'attaquant peut obtenir des encryptions de messages qu'il a lui-même choisis.*

Exercice 35. *Montrer que la méthode de chiffrement de cette section n'est pas sûre contre des attaques à messages choisis.*

3.2.2 Permuter

Une autre idée intuive pour chiffrer un bloc $m \in \{0,1\}^\ell$ serait permuter ses bits avec une permutation σ de $\{1, \ldots, \ell\}$ choisie aléatoirement, i.e. $m' = (m_{\sigma(1)}, \ldots, m_{\sigma(\ell)})$.

Exercice 36. *Combien-y-a t'il de permutations de $\{1, \ldots, \ell\}$? Proposez un choix pour ℓ*

Exercice 37. *Montrer que cette méthode de chiffrement n'est pas sûre contre des attaques à messages choisis (on pourra considérer ℓ couples (message,chiffré) pour retrouver σ).*

3.2.3 Masquer et permuter

Toutes les méthodes actuelles de chiffrement par blocs (AES,DES,...) mixent les deux techniques précédentes. Elles masquent et permutent alternativement un certain nombre de fois (appliquer un masque puis une permutation est appelé un *tour*). Après un certain nombre de tours, le chiffré devient indistinguable d'un message aléatoire. Le principal intéret de ces méthodes est leur efficacité (relativement à la cryptographie asymétrique). Cependant les méthodes de chiffrement symétriques souffrent d'un problème intrinsèque. En effet, ces méthodes nécessitent qu'une clé secrète ait été échangée au préalable. Il faut donc être capable de s'échanger des messages de manière sécurisée pour pouvoir s'échanger des messages de manière sécurisée...

1. $\oplus$ représente le XOR.

Chapitre 4

Chiffrement asymétrique

Nous définirons la notion de cryptosystème à clé publique et nous proposerons deux définitions de sécurité. Nous détaillerons ensuite le fameux cryptosystème RSA.

Pré-requis. Fiches précédentes.

4.1 Intuition

Imaginons que nous connaissions une fonction publique (connue de tous) $e : M \to C$ qui soit difficile à inverser, i.e. trouver e^{-1}. Une telle fonction, dite à sens unique, serait une bonne candidate pour chiffrer/encrypter un message $m \in M$. En effet, en supposant e^{-1} inconnue, $c = e(m)$ permettrait de chiffrer m car m serait difficile à retrouver à partir de c. Cependant, pour que le déchiffrement soit possible, il est necessaire de supposer l'existence d'une *trappe* t, à savoir une petite information auxiliaire, dont la connaissance rend e facile à inverser. Ces fonctions à sens unique, dites à trappe, sont à la base de la plupart des primitives cryptographiques. Cependant, leur existence reste une question ouverte dépendant de la question $P \neq NP$ (condition seulement nécessaire à leur existence). Ainsi, supposons qu'Alice réussisse à construire un couple (e, t). Si elle publie e et garde t secrète, Bob peut lui envoyer lui faire parvenir un message confidentiel m en lui envoyant $c = e(m)$ que seule Alice pourra déchiffrer. Pour le dire rapidement, l'objet de la cryptographie à clé publique est justement de trouver une façon efficace de construire, de manière aléatoire, des couples (e_{pk}, t_{sk}) indexées par des paires de clés (pk, sk), pk comme *public key* et sk comme *secret key*, choisies aléatoirement.

4.2 Cryptographie à clé publique

La cryptographie à clé publique (ou asymétrique) est une invention relativement récente (1978). Elle a révolutionné la cryptographie et a permis d'élargir ses ambi-

tions. Elle permet en premier lieu de solutionner le problème d'échange de clés de la cryptographie symétrique.

L'idée est de générer en même temps une clé publique pk une clé privée sk à l'aide d'une fonction $KeyGen$ La clé est ensuite publiée (par exemple sur une page web). Tout possesseur de la clé publique pk pourra encrypter alors que seul le possesseur de la clé privée sk pourra décrypter. L'intéret est que la clé privée ne passe jamais sur le réseau. Un cryptosystème Π est donc la donnée de 3 fonctions :

1. $KeyGen(\lambda)$. Comme expliqué dans la fiche précédente, le paramètre de sécurité λ permet de fixer la taille des clés. Cette taille est choisie de manière à ce chaque attaque requiert au moins 2^λ opérations élémentaires. Cette fonction renvoie un couple
$$(pk, sk)$$

2. $Encrypt(pk, m \in M)$. Algorithme polynomial qui prend en entrée un message $m \in M$ et qui renvoie une encryption $c \in C$. Cette fonction peut être probabiliste.

3. $Decrypt(sk, c \in C)$. Algorithme déterministe polynomial qui prend en entrée une encryption $c \in C$ et qui renvoie un message $m \in M$.

Un cryptosystème Π est correct si évidemment on retrouve le message encrypté en décryptant c'est à dire que les fonctions $\Pi.Encrypt$ et $\Pi.Decrypt$ sont inverses l'une de l'autre.

Définition 3. ***(Correction).*** *Un cryptosystème à clé publique est dit correct si pour tout $m \in M$,*
$$\Pi.Decrypt(sk, \Pi.Encrypt(pk, m)) = m$$

Intuitivement, on dira qu'un cryptosystème est sûr si un attaquant ne peut pas retrouver le message encrypté. Cette définition vague recouvre en fait plusieurs notions de sécurité étudiées en détail dans la section suivante.

La cryptographie à clé publique permet de sécuriser les échanges d'information sans le problème de l'échange de clés préalable. La seule raison pour laquelle la cryptographie symétrique est encore utilisée est son efficacité. Encrypter reste environ 1000 fois plus rapide avec la cryptographie symétrique, l'échange de clés pouvant se faire avec la cryptographie asymétrique. Nous verrons que la cryptographie asymétrique a élargi les ambitions de la cryptographie, i.e. vote électronique, signature numérique, etc.

Exercice 38. *Combien de paires de clés (pk, sk) doivent-être générées pour que t parties puissent communiquer de manière sécurisée ? Le comparer à celui nécessaire avec la cryptographie symétrique.*

Exercice 39. *Proposer un attaque "homme du milieu". Autrement dit, comment Eve (attaquante active sur le réseau) pourrait nuire à la communication entre Alice et Bob ?*

Infrastructure à clé publique (ICP ou PKI en anglais). L'exercice précédent montre comment l'homme du milieu peut couper la communication entre Alice et Bob

en s'interposant. En particulier, lorsque Bob souhaite récupérer la clé publique d'Alice, il peut la remplacer par la sienne. Il faut donc une infrastructure permettant de lier des clés publiques à des identités. Une telle structure est appelée *infrastructure à clé publique (ICP)*. Dans toute la suite du cours, on supposera qu'Alice (resp. Bob) connait de manière certaine, la clé publique de Bob (resp. Alice). On verra que cette condition est suffisante pour assurer la sécurisation d'un canal de communication avec la cryptographie à clé publique. Pour atteindre cette sécurisation, nous verrons au chapitre prochain la notion de signature numérique.

Limites de la cryptographie. Il n'y pas de cryptographie si l'on ne peut pas conserver, en local, une infomation secrète. Il faut donc se prémunir du risque d'intrusion. La cryptographie n'est donc qu'une composante de la sécurité informatique.

4.3 Sécurité sémantique

Il s'agit ici de définir ce que l'on entend par sécurité. La première condition est évidemment que la clé privée ne puisse pas (en temps raisonable) être retrouvée à partir de la clé publique. C'est le premier niveau de sécurité qui n'est malheureusement pas suffisant. Pour définir un niveau de sécurité, il faut définir les pouvoirs de l'attaquant (qui doivent surestimés les pouvoirs réels). Ainsi un cryptosystème est sûr contre un type d'attaquant si l'attaquant ne peut retrouver le message encrypté. Dans les deux niveaux de sécurité proposés, les attaquants auront des *a priori* différents sur le message encryptés.

•. **Sécurité sens unique.** *C'est le niveau de sécurité le plus bas où l'attaquant n'a aucun a priori sur le message encrypté. Le maitre du jeu (celui qui veut prouver que son cryptosystème est sûr) choisit un message $m \in M$ aléatoirement, génère $c \leftarrow \Pi.Encrypt(pk, m)$ et envoie c à l'attaquant. Si l'attaquant ne parvient pas à retrouver m avec une probabilité non négligeable (i.e. polynomial en $1/\lambda$), alors on parle de sécurité sens unique.*

Il est communément admis que RSA possède ce niveau de sécurité. Cependant, ce niveau de sécurité n'est pas suffisant en pratique car il est rare de pouvoir garantir que le message ait été choisi avec une probabilité uniforme sur M. De ce fait, RSA n'est pas utilisé dans sa version de base. Essayons de proposer un niveau de sécurité *plus fort* qui soit accessible en pratique.

En cryptographie asymétrique, la sécurité inconditionnelle ou parfaite au sens de Shannon n'est pas possible. La notion de sécurité sémantique (ou sécurité polynomiale) est qu'un attaquant ne puisse extraire aucune information en temps polynomial sur un message clair à partir de l'un des chiffrés, en dehors de celles qu'il aurait pu obtenir sans ce chiffré (notion introduite par Goldwasser et Micali en 1984). Il a été montré que ceci est équivalent à la définition suivante.

•. **Sécurité sémantique.** *Dans ce niveau de sécurité, l'attaquant à un a priori fort sur le message encrypté. Concrétement, le maitre du jeu choisit 2 messages $m_0, m_1 \in M$ et les envoie à l'attaquant. Il (le maitre du jeu) choisit ensuite $m \in \{m_0, m_1\}$*

aléatoirement, génère $c \leftarrow \Pi.Encrypt(pk, m)$ *et envoie* c *à l'attaquant. Si l'attaquant ne parvient pas à retrouver* m *avec une probabilité significativement plus grande que 1/2 alors on parle de sécurité sémantique.*

Autrement dit, avec ce niveau de sécurité, une encryption c ne divulgue aucune information, accessible en temps polynomial, sur le message encrypté.

Exercice 40. *Montrer que la fonction* $\Pi.Encrypt$ *doit être probabiliste pour pouvoir garantir la sécurité sémantique.*

Exercice 41. *Proposez une méthode permettant de randomiser la fonction d'encryption de RSA (idée. ajouter de l'aléatoire au message).*

Exercice 42. *Alice a généré une paire de clé* $(pk, sk) \leftarrow \Pi.KeyGen(\lambda)$ *et pose une question publique binaire (réponse oui ou non) à Bob. Bob encrypte sa réponse avec* $\Pi.Encrypt$ *et l'envoie à Alice. Quelle niveau de sécurité doit posséder* Π *?*

D'autres notions de sécurité (non étudiées ici) permettent, par exemple, à l'attaquant d'avoir accès à un oracle de décryption lui permettant de tout décrypter sauf l'encryption challenge.

4.4 RSA

RSA est le premier, le plus connu et le plus utilisé des cryptosystèmes à clé publique. Sa sécurité repose sur le problème de la factorisation. Ce cryptosystème est basé sur deux résultats vus précédemment :

- $x^{1+k\phi(n)} \equiv x \pmod{n}$
- Retrouver $\phi(n)$ à partir de n est aussi difficile que de factoriser n.

Définition 4. *Le cryptosystème à clé publique RSA est défini par :*

- $KeyGen(\lambda)$. *Générer aléatoirement deux nombres premiers* p, q *de* k *bits*[1]. *Soit* $n = pq$ *et* $\phi = (p-1)(q-1)$. *Choisir* e *arbitrairement tel que* $pgcd(e, \phi) = 1$ *et soit* d *l'inverse de* e *dans* $\mathbb{Z}_\phi$.

$$pk = (n, e); sk = d$$

- $Encrypt(pk, m \in \mathbb{Z}_n^*)$. *Retourne* $c = m^e \mod n$.
- $Decrypt(sk, c \in \mathbb{Z}_n^*)$ *Retourne* $m = m^d \mod n$.

Commençons par prouver que RSA est correct. Ceci est basé sur le théorème de Fermat-Euler.

1. k est choisi de manière à ce que le meilleur algorithme de factorisation s'execute (en moyenne) en au moins 2^λ opérations élémentaires sur n.

Proposition 9. *RSA est correct. Autrement dit, pour tout* $x \in \mathbb{Z}_n^*$

$$RSA.Decrypt(sk, RSA.Encrypt(pk, m)) = m$$

Démonstration. $(x^e)^d \mod n = x^{ed} \mod n = x^{ed \mod \phi(n)} = x^1 \mod n = x.$ □

Exercice 43. *Montrer que e ne peut pas être choisi à 2.*

Exercice 44. *Justifier le fait que RSA ne peut pas être sémantiquement sûr.*

Il n'a pas été prouvé que RSA est sûr (sécurité sens unique) si le problème de la factorisation est difficile. Celà reste un problème ouvert. Cependant, la réduction présentée dans le chapitre précédent permet de montrer que l'attaquant ne peut pas retrouver $\phi(n)$ si la factorisation est un problème difficile. Ce résultat peut être étendu pour montrer que $sk = d$ ne peut pas être retrouvé à partir de $pk = (n, e)$ si la factorisation est un problème difficile. Enfin, après 40 ans d'efforts, personne n'a encore réussi à casser la sécurité sens unique RSA. L'ordinateur quantique y parviendra peut-être !

Cependant le sécurité sens unique de RSA n'offre pas beaucoup de garanties. Il est donc facile de trouver des situations où ce niveau de sécurité ne suffit pas. De plus, des failles surviennent si des libertés sont prises par rapport aux préconisations du cryptosystèmes.

Exercice 45. *Supposons que* $e = 3$ *et que* $m < n^{1/3}$*. Expliquer comment retrouver m à partir de* $c \leftarrow RSA.Encrypt(pk, m)$*.*

Exercice 46. *Générer des grands nombres premiers est couteux. Montrer qu'il est dangereux de reutiliser des nombres premiers dans plusieurs clés publiques.*

Chapitre 5

Signature numérique

Dans le chapite précédent nous avons vu comment sécuriser les transmissions face à des attaquants passifs qui se contentent *d'écouter* le réseau. Cependant, ceci n'est pas suffisant face à des attaquants type *homme du milieu* qui pourraient modifier les messages. En effet, imaginons qu'Alice envoie un message crypté à c à Bob. Rien n'empêche Charles de subsituter à c, une encryption c' du message de son choix. Il faut donc trouver un moyen de garantir à Bob que le message reçu a bien été envoyé par Alice et qu'il n'a pas été modifié au cours de la transmission. La signature numérique propose de solutionner ce problème. Cette primitive cryptographique peut être vue comme duale au chiffrement asymtétrique. Il est à noter que la signature manuscripte n'offre pas les mêmes garanties puisqu'elle peut être imitée et réutilisée.

5.1 Principe

5.1.1 Cahier des charges

Nous pouvons citer quelques propriétés que devrait vérifier une la signature numérique :

1. *Authentique* : l'identité du signataire doit pouvoir être retrouvée de manière certaine.
2. *Infalsifiable* : la signature ne peut pas être falsifiée. Quelqu'un ne peut se faire passer pour un autre.
3. *Non réutilisable* : la signature n'est pas réutilisable. Elle fait partie du document signé et ne peut être déplacée sur un autre document.
4. *Inaltérable* : un document signé est inaltérable. Une fois qu'il est signé, on ne peut plus le modifier.
5. *Irrévocable* : la personne qui a signé ne peut le nier.

De ces propriètés informelles, nous pouvons déja tirer quelques premières conclusions :

Conclusion 6. *La propriète 1 et la propriète 3 implique que la signature doit dépendre de l'expéditeur de du message lui-même (contrairement à la signature manuscripte qui ne dépend que de l'expéditeur).*

5.1.2 Formalisation

Munissons nous d'un ensemble de messages I_m, d'un ensemble de signatures I_s et d'un ensemble de paires de clés I_k.

Définition 5. *Un procédé de signature est la donnée pour chaque $(pk, sk) \in I_k$, de deux fonctions complexité polynomiale :*

1. *$sig_{sk} : I_m \to I_s$: fonction de signature (secrète)*
2. *$ver_{pk} : I_m \to I_s \times \{vrai, faux\}$: la fonction de vérification (publique) telles que*

$$ver_{pk}(m, \sigma) = vrai \text{ si } \begin{cases} vrai, & \sigma = sig_{sk}(m) \\ faux, & sinon \end{cases}$$

Il faut tout d'abord remarquer qu'un procédé de signature n'est jamais inconditionnellement sûr (c'est à dire que la sécurité n'est pas garantie si Alice est capable de calculer des fonctions de complexité non-polynomiale, e.g. elle possède un ordinateur quantique). En effet, Alice, désirant signer un message m à la place de Bob, peut essayer une à une toutes les valeurs de $\sigma \in I_s$ et trouver une signature valide grâce à la fonction de vérification ver_{pk}.

5.2 Système de signature basé sur RSA

Un système de signature peut être vu comme un système dual à la cryptographie à clé publique. Nous présentons ici la signature basée sur RSA. L'idée intuitive est d'utiliser la clé secrète pour signer et la clé publique pour la vérification. Une première idée (naive) serait d'encrypter le message avec la clé privée. Mais cette idée n'est pas du tout pertinente car il est facile de produire des contrefaçons. En effet, tout couple $(\sigma^e, \sigma) \in (\mathbb{Z}/n\mathbb{Z})^2$ passe la vérification.

Pour dépasser ceci, nous allons introduire l'utilisation d'une table de hachage publique $\mathcal{H} = \{0,1\}^* \to \{0,1\}^\ell$ et nous supposerons qu'Alice a généré $(pk = (n, e), sk = d) \leftarrow RSA.KeyGen(\lambda)$. Pour signer un message, Alice n'encryptera pas, avec sa clé privée, le message lui-même mais le haché $\mathcal{H}(m)$ de ce message (on supposera que $2^\ell < n$). Plus concrètement :

1. sig_{sk}. Générer $h \leftarrow \mathcal{H}(m)$ et retourner $\sigma = h^d \mod n$
2. ver_{pk}. la fonction de vérification (publique) telles que

$$ver_{pk}(m, \sigma) = \text{vrai si } \begin{cases} vrai, & \sigma^e = \mathcal{H}(m) \\ faux, & sinon \end{cases}$$

Exercice 47. *Montrer que la fonction de vérification est correct.*

Pour que ce système de signature soit sûr, il faut que la table de hachage $\mathcal{H}$ possède des prorpriètés particulières (on parlera de tables de hachages cryptographiques). Plus clairement, elle doit être résitantes aux collisions. Ceci recouvre plusieurs définitions possibles.

Définition 6. *Soit $\mathcal{H}$ une table de hachage.*

1. *$\mathcal{H}$ est dite **faiblement résistante** aux collisions si étant donné m_0, il est calculatoirement difficile d'obtenir m tel que $\mathcal{H}(m) = \mathcal{H}(m_0)$.*
2. *$\mathcal{H}$ est dite **fortement résistante** aux collisions s'il est calculatoirement difficile d'obtenir m et m' distincts tels que $\mathcal{H}(m) = \mathcal{H}(m')$.*

Evidemment, une table de hachage fortement resitante aux collisions l'est aussi faiblement. Une attaque découle immédiatement si $\mathcal{H}$ n'est pas faiblement résistante aux collisions. En effet, si Charles intercepte un couple (m, σ), il peut le remplacer par le couple (m', σ) vérifiant $\mathcal{H}(m') = H(m)$. En effet, σ est aussi une signature valide de m'. Cependant, la faible résistance aux collisions n'est pas suffisante.

Exercice 48. *Proposez une attaque sur le système de signature RSA si $\mathcal{H}$ n'est pas fortement resistante aux collions.*

Il reste maintenant à trouver des tables de hachages cryptographiques qui sont des primitives incontournables en cryptographie. Nous en proposons une dont la résistance aux collisions sous réserve que le logarithme discret soit un problème difficile (retrouver $x = \log_\alpha(\beta)$, connaissant $\beta = \alpha^x \mod n$).

Définition 7. ***(table de hachage de Chaum et al.)*** *Soit p un nombre premier tel que $q = (p-1)/2$ soit également premier. Soit α et β deux éléments primitifs modulo p. On suppose qu.il est difficile de calculer $\log_\alpha(\beta)$: la valeur x telle que $\alpha^x = \beta \mod p$. On définit la fonction de Chaum-Van Heist-Pfitzmann $h : \{0, \ldots, q-1\}^2 \to \mathbb{Z}_p$ par*

$$h(x, y) = x^\alpha y^\beta \mod p$$

Proposition 10. *Toute collision dans la fonction ci-dessus permet de calculer $log_\alpha(\beta)$.*

Démonstration. Supposons que l'on a une collision, c'est à dire deux couples distincts (x, y) et (x', y') tel que $\alpha^x \beta^y = \alpha^{x'} \beta^{y'} \Rightarrow \alpha^{x-x'} = \beta^{y'-y}$ dans $\mathbb{Z}/p\mathbb{Z}$. Soit $d = pgcd(p-1, y'-y)$. Comme $p-1 = 2q$ avec q premier et $-q < y'-y < q$, $d = 1$ ou $d = 2$. Examinons ces 2 cas :

— $d = 1$. Dans ce cas, $y - y'$ est inversible modulo $p-1$ et soit r son inverse. On a donc $\alpha^{r(x-x')} = \beta$ ce qui signifie que $\log_\alpha(\beta) \equiv r(x - x') \pmod p - 1$.

— $d = 2$. Soit r l'inverse de $y - y'$ modulo q. Donc, $r(y' - y) = 1 + kq$ ce qui implique que $\alpha^{r(x-x')} = \beta^{1+kq} = \beta \cdot (\beta^q)^k$. Comme $\beta^q = -\beta$ (car β est primitif et $\beta^{p-1=2q} = 1$. Donc $\log_\alpha(\beta)$ est soit égal à $r(x-x') \mod p-1$ soit à $r(x-x')+q \mod p-1$

□

Exercice 49. *Proposez une table de hachage cryptographique (résistante aux collisions) utilisant la fonction de Chaum-Van Heist-Pfitzmann.*

Cette fonction de hachage n'est cependant pas utilisée en pratique, car très coûteuse. D'autres tables de hachage sont utilisées en pratique (MD5,SHA-1,SHA-2) bien plus efficaces même si leur resistance aux collisions n'est pas prouvée.

Chapitre 6

Cryptosystèmes homomorphiques

On supposera que l'ensemble des messages clairs M des cryptosystèmes de cette section est $M = \mathbb{Z}/n\mathbb{Z}$.

6.1 Sécuriser le cloud

La sécurité dans le cloud est un des enjeux importants en informatique. Il s'agit pour un utilisateur d'externaliser ses données pour qu'elles soient disponibles en ligne. Le problème de la confidentialité se pose immédiatement. Une solution serait de ne pas mettre en ligne les données elles-mêmes mais des encryptions de ces données. Cette solution n'est toutefois pas totalement satisfaisante. En effet, lorsqu'un utilisateur veut manipuler ses données, il doit au préalable les télécharger et les décrypter. Ceci peut s'avérer inopérant car trop coûteux. Il faudrait aussi pouvoir externaliser les calculs. Serait-il possible de faire en sorte que les calculs puissent être effectués dans le cloud sur le données encryptées ? Plus précisément, serait-il possible que le cloud puisse obtenir une encryption du résultat qui serait envoyé et décrypté en local par l'utilisateur. Les cryptosystèmes homomorphiques permettent de réaliser ce miracle.

6.2 Qu'est ce qu'un cryptosystème homomorphique ?

Considérons le tour de magie suivant. Vous choisissez deux nombres m_1, m_2 entre 0 et 10. Vous écrivez ensuite ces deux nombres sur deux morceaux de papier. Chacun de ces deux morceaux de papier sera enfermé dans une boite dont vous garderez la clé. Vous donnez ces deux boites au magicien ainsi que 21 autres boites contenant chacune un nombre secret compris entre 0 et 20 (deux boites différentes contiennent deux nombres différents). Le magicien devra choisir, avec succès, la boite qui contient $m_1 + m_2$. Vous ouvrirez la boite pour vérifier. C'est ce tour de magie bluffant que permettent de réaliser les cryptosystèmes homomorphiques. Soit $\Pi = (KeyGen, Encrypt, Decrypt)$ un cryptosystème à clé publique.

Cryptosystème additivement homomorphique : $\oplus, \circ$

Le cryptosystème Π est additivement homomorphique s'il existe une fonction publique[1] $\oplus : C \times C \to C$ vérifiant la propriété suivante :

$$\Pi.Decrypt(sk, \oplus(c, c')) = \Pi.Decrypt(sk, c) + \Pi.Decrypt(sk, c')$$

$+$ étant l'addition sur $\mathbb{Z}/n\mathbb{Z}$. Il est à noter qu'on peut construire un opérateur $\circ : \mathbb{N} \times C \to C$, à partir de $\oplus$, qui permet de contruire une encryption $c' = a \circ c$ de ax connaissant a et une encryption c de x, i.e.

$$a \circ c = \underbrace{c \oplus c \oplus \cdots \oplus c}_{a \text{ times}}$$

Exercice 50. *Montrer que $a \circ c$ peut être calculé efficacement en $O(\log a)$ (en faisant comme dans ExpMod).*

Cryptosystème multiplicativement homomorphique : $\otimes$

Tout pareil que les crptosystèmes additivement homomorphiques sauf que l'on considère une fonction publique $\otimes : C \times C \to C$ tel que

$$\Pi.Decrypt(sk, \otimes(c, c')) = \Pi.Decrypt(sk, c) \times \Pi.Decrypt(sk, c')$$

Cryptosystème (pleinement) homomorphique.

Un cryptosystème homomorphique est un cryptosystème à la fois additivement et multiplicativement homomorphique. On montre que si l'on dispose d'un tel cryptosystème toute fonction peut être évaluée de manière homomorphique. *Il est important de comprendre que les opérations homomorphiques sont publiques et ne requierent pas la connaissance de la clé privée.*

Proposition 11. *Soient $c_1, \ldots, c_t$ des encryptions de $x_1, \ldots, x_t \in \{0, 1\}$ et f une fonction booléenne arbitraire. On peut obtenir une encryption de $f(x_1, \ldots, x_t)$ en n'utilisant que les propriétés homomorphiques de Π (et donc sans utiliser la clé privée).*

Exercice 51. *Prouver la proposition 11.*

6.3 Applications

6.3.1 Sécuriser le cloud

L'application est immédiate grâce à la proposition 11.

1. C désigne l'ensemble des encryptions.

6.3.2 Vote électronique

Imaginons qu'il y aient deux candidats Alice et Bob et t votants. Si le votant i choisit de voter pour Alice, il choisit par convention d'envoyer une encryption c_i de 0 (et de 1 sinon). Les propriétés homomophiques permettent d'obtenir $c = c_1 \oplus \cdots \oplus c_t$. Pour obtenir le résultat du vote, il suffit de calculer $x \leftarrow Decrypt(sk, c)$. Si $x < n/2$ alors Alice est élue sinon c'est Bob. On a donc réussi a obtenir le résultat du vote tout en préservant la confidentialité des votes individuels (les encryptions c_i n'ont pas été décryptées).

Exercice 52. *Pourquoi faut-il que le cryptosystème utilisé pour le vote électronique soit sémantiquement sûr ?*

6.3.3 Requête secrète

Supposons que A dispose d'une base de données secrète BD. Bob souhaiterait requêter BD (obtenir une des valeurs de BD) tout en gardant sa requête secréte. Ceci peut être réalisé avec un cryptosystème homomorphique.

Exercice 53. *Voir exercice 3 de la section 8.5.*

6.3.4 Autres

Beaucoup d'autres applications existent, notamment en machine learning. Il s'agit de construire des classifieurs sur des bases de données distribuées tout en préservant leur confidentialité.

6.4 Exemples de cryptosystèmes homomorphiques

De nombreux cryptosystèmes additivement ou multiplicativement homomorphiques ont été développés. Il est à noter que RSA est multiplicativement homomorphique.

Exercice 54. *Montrer que RSA est multiplicativement homomorphique.*

Ce n'est que très récemment (2009) que des cryptosystèmes homomorphiques (à la fois additivement et multiplicativement) ont vu le jour. Bien que beaucoup de progrès ont été réalisés, ces outils restent encore largement inefficaces. Un exemple sera donné en fin de section.

6.4.1 Cryptosystème de Paillier

Le cryptosystème de Paillier (1999) est le cryptosystème additivement homomorphique réunissant beaucoup de *bonnes propriétés*. Il permet d'encrypter sur $\mathbb{Z}/n\mathbb{Z}$, il est probabiliste et permet de randomiser une encryption. Ceci sera explqué ci-dessous.

Définition 8. *Le cryptosystème à clé publique Paillier est défini par :*

— *KeyGen*(λ). *Générer aléatoirement deux nombres premiers* p, q *de* k *bits*[2]. *Soit* $n = pq$ *et soit* $\rho = n^{-1} \mod \phi$.

$$pk = n; sk = \rho$$

— *Encrypt*($pk, m \in \mathbb{Z}_n^*$). *Choisir aléatoirement* $r \in \mathbb{Z}/n\mathbb{Z}$. *Retourne*

$$c = (1 + mn)r^n \mod n^2$$

— *Decrypt*($sk, c \in \mathbb{Z}_n^*$). *Calcule* $r = (c \mod n)^\rho$. *Retourne*

$$m = \frac{(c \times r^{-n} \mod n^2) - 1}{n}$$

On remarque qu'une encryption $c \in \mathbb{Z}/n^2\mathbb{Z}$ et est de taille (nombre de bits) 2 fois supérieure au message encrypté. De plus, le temps d'encryption et de décryption est significativement plus important qu'avec RSA. Nous allons cependant voir que ce cryptosystème possède des propriétés intéressantes que RSA ne possède pas.

Exercice 55. *Prouver que le cryptosystème de Paillier est correct.*

Proposition 12. *Le cryptosystème de Paillier est* ***additivement homomorphique*** *tel que* $c \oplus c' = cc' \mod n^2$ *et donc* $a \circ c = c^a \mod n^2$.

Démonstration. Soit c, c' deux encryptions de respectivement $m, m' \in \mathbb{Z}/n\mathbb{Z}$.
$cc' \mod n^2 = (1+nm)r^n(1+nm')r'^n \mod n^2 = (1+n(m+m')+n^2(rr')^n \mod n^2 = (1+n(m+m')r''^n$ avec $r'' = rr'$. C'est donc une encryption de $m + m'$. □

Voici d'autres propriétés fondamentales de ce cryptosystème.

1. **Sécurité sémantique.** Le cryptosystème de Paillier est probabiliste (r est un nombre choisi aléatoirement lors de chaque encryption). On peut donc espérer atteindre le niveau de sécurité sémantique. Celle-ci est démontrée en supposant que la factorisation est difficile (en fait, la sécurité sémantique de Paillier est liée à un problème connexe à la factorisation, la n-residuosity).

2. **Randomisation des encryptions.** Les encryptions obtenues en utilisant les propriétés homomorphiques peuvent être randomisées. Soit z une encryption de 0 générée avec *Paillier.Encrypt*. l'encryption $c'' = c \oplus c' \oplus z$ est aussi une encryption $m + m'$. Cependant, conaissant (c, c', c''), l'attaquant ne voit aucun lien entre c, c' et c'' et ne sait pas en particulier que c'' encrypte la somme des valeurs encryptées par c et c'. Autrement dit, une encryption randomisée est indistinguable d'une encryption fraiche (encryptant le même message) renvoyer par la fonction *Encrypt*.

2. k est choisi de manière à ce que le meilleur algorithme de factorisation s'execute (en moyenne) en au moins 2^λ opérations élémentaires sur n.

3. **Preuve de décryption.** Soit c une encryption arbitraire connu de Bob et de Alice qui a généré $(pk, sk) \leftarrow Paillier.KeyGen(\lambda)$. Alice peut donc décrypter c, i.e. calculer $m \leftarrow Paillier.Decrypt(sk, c)$ et prouver à Bob que c est une encryption de c. En effet, lors de la décryption, Alice a retrouvé la valeur aléatoire r. Elle peut donc envoyer (m, r) à Bob. Pour vérifier que la décryption est correct, Bob doit vérifier que $c = (1 + mn)r^n \mod n^2$.

Exercice 56. *Soient deux messages m_0, m_1 publiques. Supposons qu'Alice choisisse aléatoirement un bit b, génère une encryption M de m_b, i.e. $M \leftarrow Paillier.Encrypt(pk, m_b)$ et l'envoie à Bob. Supposons que Charles contrôle le réseau mais ne connait pas la clé privée sk associée à pk. Peut-il remplacer M par une encryption M' de m_{1-b} ?*

6.4.2 Cryptosystème de Van Dijk et al.

Le cryptosystème de Van Dijk est un cryptosystème homomorphique (2010) très simple bien que pas très performant. Il permet d'encrypter un bit $m \in \mathbb{Z}/2\mathbb{Z}$. Pour cela, un grand entier secret p est choisi. Une encryption c de m est simplement un entier qui est presque un multiple de p, i.e.

$$c \leftarrow kp + 2e + m$$

avec e est un entier choisi aléatoirement tq $e \ll p$. Vérifions les propriétés homomorphiques

— $c + c' = kp + 2e + m + k'p + 2e' + m' = (k + k')p + 2(e + e') + m + m'$. C'est donc une encryption de $m + m'$ sous réserve que $2(e + e') < p - 2$

— $cc' = (kp+2e+m)(k'p+2e'+m') = (kk'p+2ek'+2e'k)p+2(2ee'+em'+e'm)+mm'$. C'est donc une encryption de mm' sous réserve que $2(2ee'+em'+e'm) \leq 2(2ee'+ e + e') < p - 1$.

Exercice 57. *Vérifiez que pour décrypter c, il suffit de calculer $(c \mod p) \mod 2$*

Exercice 58. *Vérifier les propriètés homomorphiques sur des exemples simples (p petit).*

Exercice 59. *Le chiffrement a été présenté de manière symétrique car p est utlisé pour chiffrer et déchiffrer. Comment rendre ce cryptosystème asymétrique en utilisant ces propriétés homomorphiques (additives) ?*

Exercice 60. *Proposer une attaque, eventuellement par force brute.*

Exercice 61. *Montrer que si c_0 est une encryption de 0, alors c' et $c' \mod c_0$ encryptent la même valeur. Comment utiliser ceci pour la taille des encryptions ne grossissent avec les opérations homomorphiques.*

6.5 Limites des cryptosystèmes homomorphiques

La proposition 11 laisse à penser que les cryptosystèmes homomorphiques résolvent le problème de la sécurité dans le cloud. Ceci est à relativiser fortement. En effet, n'importe quelle fonction f peut être évaluée homomorphiquement. Cependant, ce n'est efficace que si f peut s'écrire efficacement comme un circuit arithmetique. Ceci n'est notamment pas le cas si cette fonction contient beaucoup des tests (**si-alors**). Pour dépassser ceci, les chercheurs travaillent sur des techniques d'offuscation dont le but de masquer le déroulement d'un algorithme. Bien que quelques succès existent, il y a des résultats généraux d'impossibilité.

Chapitre 7

Calcul multi-parties sécurisé

Nous avons déjà evoqué des exemples de calcul multi-parties dans les chapitres précédents, e.g. le vote électronique. Nous nous proposons ici de formaliser cette notion et d'analyser quelques exemples. Dans ce chapitre, nous ferons l'hypothèse que les canaux de communications sont sécurisés (avec les cryptosystèmes et systèmes de signature numérique vus précédemment).

7.1 Formalisation

Supposons la présence de T parties $P_1, \ldots, p_T$ disposant chacune de données secrètes, i.e. P_i connait x_i qu'elle souhaite conserver secrète. Ces parties souhaitent collaborer de manière à ce qu'à la fin de leur collaboration, appelé protocole, la partie P_1 ait reçu $f_1(x_1, \ldots, x_T)$, la partie P_2 ait reçu $f_2(x_1, \ldots, x_T)$, etc. où $f_1, f_2, \ldots, f_T$ sont des fonctions connues de tous. Il doit aussi être assuré que chaque partie P_i n'a reçu **que** $f_i(x_1, \ldots, x_T)$ et rien d'autre d'informatif sur les données secrètes x_j des autres parties.

Exemple 7. *Deux millionnaires A et B souhaitent collaborer pour savoir qui est le plus riche sans rien connaitre d'autre sur la fortune de l'autre.*

Exemple 8. *Une classe d'étudiants souhaite connaitre la moyenne des notes de crypto sans que soient dévoilées les notes individuelles.*

Pour simplifier notre propos, nous nous limiterons au cas $T = 2$. Alice et Bob souhaitent donc interagir pour obtenir $f_A(x_A, x_B)$ et $f_B(x_A, x_B)$. Comme nous avons supposé que les canaux de communication sécurisés, nous pouvons considérés qu'ils sont isolés du reste du monde. Ainsi, Alice et Bob n'ont rien à redouter du monde extérieur mais sont tous les deux concurrents. Alice pourrait souhaiter obtenir $f_A(x_A, x_B)$ mais que Bob obtienne une valeur erronée. Un protocole sera dit sûr si évidemment çela, par exemple, n'est pas possible. Mais comment définir formellement la sécurité d'un protocole ? Pour cela, il faut définir un cas idéal auquel se référer.

7.2 Cas ideal

Dans le cas ideal, on suppose l'existence d'une tierce partie de confiance $\mathfrak{T}$. La partie de confiance est supposée ne pas être corrompue et respecter le protocole. Le protocole "*ideal*" se déroule de la manière suivante :

1. Alice envoie x_A à $\mathfrak{T}$
2. Bob envoie x_B à $\mathfrak{T}$
3. $\mathfrak{T}$ calcule $y_A = f_A(x_A, x_B)$ et $y_B = f_B(x_A, x_B)$
4. $\mathfrak{T}$ envoie y_A à Alice et y_B à Bob.

Ce protocole garantit qu'Alice et Bob reçoivent des valeurs correctes et que x_A et x_B restent confidentielles. A l'issue de protocole, Alice *n'aura appris* que $f_A(x_A, x_B)$ et Bob que $f_B(x_A, x_B)$. De plus, Bob et Alice auront choisi leur données de manière indépendante. Tout l'enjeu du calcul multi-partie sécurisé est de construire des protocoles offrant les mêmes garanties mais sans la partie de confiance $\mathfrak{T}$.

Il est à noter, que dans le cas idéal, les parties peuvent choisir les données qu'elles souhaitent. La cryptographie ne peut pas rien faire contre ça. Par exemple, dans le cas des millionnaires (voir exemple 7), les millionnaires peuvent mentir sur leur fortune.

Personne ne peut obliger une partie à participer au protocole dans le cas idéal. Nous sommes aussi obligé d'ajouter un pouvoir (inélégant) à une des deux parties (disons Alice). Cette partie peut, après avoir obtenu $f_A(x_A, x_B)$, arrêter le protocole de manière à ce que Bob ne reçoive rien. Malheureusement, ceci doit être ajouter au cas idéal. En effet si on ne le faisait pas, aucun protocol réel ne pourrait être sûr car il a été démontré qu'il est impossible d'empêcher une des deux parties de tirer advantage d'un retrait prématuré du protocole.

7.3 Cas réel

Etant donné une fonctionalité $f = (f_A, f_B)$, il s'agit de construire un protocole P implémentant f en offrant les mêmes garanties que le cas idéal. Nous supposerons qu'au moins une des deux parties est honnête, disons Alice, à savoir qu'elle respecte le protocole (si les deux parties sont malhonnêtes, le protocole n'est pas sensé garantir quoi que ce soit...encore heureux). On peut néanmaoins définir plusieurs niveaux de malhonnêteté pertinents :

— **Malhonnêteté passive.** La partie malhonnête respecte le protocole. Cependant elle est curieuse (elle peut faire des calculs pour essayer d'obtenir des informations supplémentaires par exemple).

— **Malhonnêteté active.** La partie malhonnête agit arbitrairement (elle peut dévier du protocole comme elle l'entend).

— Des cas intermédiaire peuvent être considérés. Par exemple, une partie malhonnête ne veut pas que sa malhonnêteté soit repérée (sinon sanctions pénales par exemple).

Il s'agit de construire un protocole offrant à Alice (resp. Bob) les mêmes garanties que dans le cas idéal quelque soit le comportement de Bob (resp. Alice).

Définition 9. ***(simplifiée).*** *Un protocole (entre deux parties) est dit sûr s'il offre les mêmes garanties que le cas idéal, autrement dit, s'il garantit à la partie honnête (ici Alice) que :*

1. *x_B a été choisie indépendament (sans avoir aucune information) de x_A.*
2. *x_A n'a pas été divulguée. Autrement dit, $f_B(x_A, x_B)$ est la seule information relative à x_A que Bob ait reçue.*
3. *Alice ne retourne pas une valeur incorrecte. Autrement dit, Alice reçoit $f_A(x_A, x_B)$ ou éventuellement ne recoit rien si Bob a quitté le protocole prématurément.*

Exercice 62. *Proposer un protocole pour la fonctionnalité (f_A, f_B) définie par $f_A(x_A, x_B) = f_B(x_A, x_B) = x_A x_B$.*

Théorème 5. *Toute fonctionalité peut être implémentée manière sécurisée (quelque que soit le comportement de la partie malhonnête).*

Ce beau résultat théorique a cependant peu d'impact en pratique car la construction générique utilisée pour la preuve de ce résultat est très couteuse.

7.4 Calcul multi-parties sécurisé vs cryptosystèmes homomorphiques

Les mesures de qualités d'un protocole sont :

1. *La complexité algorithmique*
2. *Le volume de communication*
3. *Le nombre d'échanges*

Les cryptosystèmes homomorphiques sont des outils très puissants pour réduire ce dernier critère (nombre d'échanges). En effet, il permettent de faire tous les calculs en local (sur des valeurs encryptées en utilisant les propriètés homomorphiques). Il reste à échanger les clés, les encryptions et à décrypter (+ preuve de la décryption). Des protocoles génériques montrent que tout ceci peut se faire en 4 échanges au maximum voire 2 en acceptant comme vraies des conjectures communément admises. Le revers de la médaille est que la complexité algorithmique et le volume de communication sont élevés. Des progrès significatifs restent à faire pour que ces outils (les cryptosystèmes homomorphiques) deviennent incontournables dans le calcul multi-parties sécurisé.

7.5 Exemple : le protocole Multiplication

Soit $\Pi = (KeyGen, Encrypt, Decrypt)$ un cryptosystème **additivement homomorphique** mais pas multiplicativement. On supposera que l'ensemble des messages clairs considéré par Π est $\mathsf{Z}/n\mathsf{Z}$. Supposons qu'Alice possède deux encryptions X et Y de deux valeurs inconnues x, y encryptés par $\Pi.Encrypt$. Alice ne peut pas obtenir seule (en local) une encryption Z de $z = xy$ (car Π n'est pas supposé multiplicativement homomorphique). Nous allons voir qu'elle peut cependant obtenir une telle encryption en interagissant avec le possesseur de la clé secrète sk, i.e. Bob.

Multiplication.

Pré-requis. *Bob a généré* $(pk, sk) \leftarrow \Pi.KeyGen(\lambda)$. *Alice a deux encryptions* X *et* Y *de deux valeurs inconnues* x, y.

1. Alice choisit r, s aléatoirement dans $\mathbb{Z}/n\mathbb{Z}$. Ensuite, elle encrypte r, s, i.e. $R \leftarrow \Pi.Encrypt(pk, r)$ et $S \leftarrow \Pi.Encrypt(pk, s)$. Enfin, elle envoie à Bob $U = R \oplus X$ et $V \oplus S$

2. Bob decrypte U et V, i.e. calcule $u \leftarrow \Pi.Decrypt(sk, U)$ et $v \leftarrow \Pi.Decrypt(sk, V)$. Ensuite, il encrypte et envoie à Alice le produit uv, i.e. $W \leftarrow \Pi.Encrypt(pk, uv)$.

3. Alice retourne l'encryption $W \oplus (-s \circ X) \oplus (-r \circ Y) \oplus \Pi.Encrypt(pk, -rs)$.

Proposition 13. *Le protocole* ***Multiplication.*** *est sûr contre une partie malhonnête passive en admettant que* Π *est sémantiquement sûr.*

Exercice 63. *Prouver la proposition 13 en vous servant de la définition 9.*

Cependant ce protocole n'est pas sûr contre des parties malhonnêtes actives arbitraires. En effet, Bob peut faire n'importe quoi à l'étape 2 du protocole et envoyer n'importe quoi à Alice. Il semble difficile de trouver une parade à ceci. Cependant, les cryptologues ont développés des outils fascinants pemettant de résoudre ce genre de problèmes : *Preuve intérative à divulgation nulle de connaissance*. Dans notre cas, Bob va pouvoir prouver qu'il effectue correctement la multiplication sans rien divulguer sur les valeurs encryptées. Ceci est l'objet du TP 2 qui se trouve à la fin de ce fascicule.

7.6 Exemple/exercice : le produit scalaire

Alice dispose d'un vecteur $\overrightarrow{u} = (u_1, \ldots, u_t)$ et Bob d'un vecteur $\overrightarrow{v} = (v_1, \ldots, v_t)$. On souhaite élaborer un protocole implémentant la fonctionalité $f = (f_A, f_B)$ définie par

$$f_A(\overrightarrow{u}, \overrightarrow{v}) = f_B(\overrightarrow{u}, \overrightarrow{v}) = \overrightarrow{u} \cdot \overrightarrow{v}$$

Nous proposons le protocole ProdScal suivant :

1. Alice génère $(pk, sk) \leftarrow Paillier.KeyGen(\lambda)$ et publie pk. Elle génère $c_i \leftarrow Paillier.Encrypt(pk, u_i)$ pour tout $i = 1, \dots t$ et envoie $c_1, \dots, c_t$ à Bob.

2. En utilisant les propriétés homomorphiques de Paillier (son cryptosystème surtout), Bob génère l'encryption $c = (c_1 \circ v_1) \oplus \cdots \oplus (c_t \circ v_t)$ et l'envoie à Bob.

3. Alice decrypte c, i.e. calcule $p \leftarrow Paillier.Decrypt(sk, c)$ et l'envoie à Bob.

4. Alice et Bob retournent p.

Exercice 64. *Ce protocole est-il sûr ? Si ca n'est pas le cas, proposez des améliorations.*

Chapitre 8

Quelques exercices supplémentaires

Dans ce chapitre, $\mathbb{Z}/n\mathbb{Z}$ sera noté $\mathbb{Z}_n$.

8.1 Quelques exercices d'arithmétique modulaire

Exercice 65. *Il s'est ecoulé 210476 heures depuis le 1/01/2010. Quelle heure est-il ?*

Exercice 66. *La relation $\equiv_n$ sur $\mathbb{Z}$ est définie par $a \equiv_n b$ (on pourra noter $a \equiv b \pmod{n}$) si et seulement s'il existe un entier $k \in \mathbb{Z}$ tel que $a = b + kn$. Montrez que $\equiv_n$ est une relation d'équivalence.*

Exercice 67. *Soit $a \in \mathbb{Z}$. Montrer que $a \mod n \equiv a \pmod{n}$ ($a \mod n$ étant le reste de la division de a par n).*

Exercice 68. *Supposons que $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n}$. Montrer que :*

1. $a_1 + b_1 \equiv a_2 + b_2 \pmod{n}$
2. $a_1 b_1 \equiv a_2 b_2 \pmod{n}$

Exercice 69. *Soit $a \in \mathbb{Z}$. On note $\overline{a}_n$ la classe à laquelle appartient a. Montrer que $\mathbb{Z}_n = \{\overline{0}_n, \overline{1}_n, \ldots, \overline{(n-1)}_n\}$.*

Exercice 70. *On définit les opérations sur $\mathbb{Z}_n$ suivantes :*

1. $\overline{a}_n + \overline{b}_n = \overline{(a+b)}_n$
2. $\overline{a}_n \overline{b}_n = \overline{(ab)}_n$

Montrez que ces opérations sont bien définies (remarque : on a ainsi défini une structure d'anneau sur $\mathbb{Z}_n$).

Exercice 71. *Vérifier que pour tout $x \in \mathbb{Z}_7$, $x^6 \equiv 1 \pmod{7}$. En déduire que tout élement non-nul de $\mathbb{Z}_7$ est inversible (et donc que $\mathbb{Z}_7$ est un corps).*

Exercice 72. *Montrer que $x^5 - x \equiv 0 \pmod{3}0$ n'est pas équivalent à $x^4 - 1 \equiv 0 \pmod{3}0$ ou $x \equiv 0 \pmod{3}0$. On explicitera les 2 ensembles de solution.*

Exercice 73. *Montrer que $n^5 - n$ est divisible par 30 pour tout $n \in \mathbb{N}$*

Exercice 74. *Résoudre les équations suivantes :*
- $3x \equiv 5 \pmod 7$
- $6x \equiv 9 \pmod 1 5$
- $7x \equiv 1 \pmod 1 4$

Exercice 75. *Quels sont les éléments inversibles dans $\mathbb{Z}_5$ et dans $\mathbb{Z}_6$?*

Exercice 76. *Combien y-a-t-il d'entiers n dans $\{1,\ldots,128\}$ tels que $pgcd(n, 100) = 1$?*

Exercice 77. *Vérifier que la fonction $f : \mathbb{Z}_{15} \to \mathbb{Z}_3 \times \mathbb{Z}_5$ définie par $f(x) = (x \mod 3; x \mod 5)$ est une bijection ? Quelle est la fonction inverse ?*

Exercice 78. *Combien l'armée de Han Xing comporte-t-elle de soldats si, rangées par 3 colonnes, il reste deux soldats, rangées par 5 colonnes, il reste trois soldats et, rangées par 7 colonnes, il reste deux soldats ?*

Exercice 79. *Soit $n = pq$ le produit de deux nombres premiers p et q. On définit les sous-ensembles E_p et E_q de $\mathbb{Z}_n$ par $E_p = \{0, p, \ldots, (q-1)p\}$ et $E_q = \{0, q, \ldots, (p-1)q\}$. On note que l'ensemble des éléments non-inversible de $\mathbb{Z}_n$ est égal à $E_p \cup E_q$ et $E_p \cap E_q = \{0\}$. Soit $a, b \in \mathbb{Z}_n$. Quel est le nombre de solutions de l'équation $ax + b = 0$. On distinguera les 16 cas concernant l'appartenance ou non de a et b aux ensembles E_p et E_q. En déduire que si a est non nul, i.e. $a \notin E_p$ ou $a \notin E_q$, alors le nombre de solutions est inférieur à $\max(p, q)$.*

8.2 Quelques exercices d'algorithmie...

Exercice 80. *Parmi les problèmes suivants, quels sont ceux dont on connait des algorithmes rapides (dont l'exécution prend moins d'un siècle dans le pire des cas avec les meilleurs machines actuelles sur des nombres de taille 1024 bits).*

1. *Le problème du voyageur de commerce*
2. *Trouver le plus court chemin dans un graphe.*
3. *Tester si un entier est premier*
4. *trouver un nombre premier de taille 1024 bits*
5. *Factoriser des entiers*
6. *Calculer le pgcd entre 2 entiers*
7. *Calculer $a^b \mod n$*
8. *Le probleme de la sous-somme*

Exercice 81. *Alice (toujours elle) propose l'algorithme suivant pour choisir un nombre RSA $n = pq$ de taille $k = 2k'$*

Entrée *: $k = 2k'$*

Choisir aléatoirement m *dans* $\{2^{k'}, ..., 2^{k'+1} - 1\}$

Tant que m *n'est pas premier*

$m \leftarrow m + 1$

$p \leftarrow m$

$m \leftarrow m + 1$

Tant que m *n'est pas premier*

$m \leftarrow m + 1$

$q \leftarrow m$

Renvoyer $n = pq$

---

1. *Cet Algorithme est-il correct ? En particulier renvoit-il toujours un nombre de taille* k *?*
2. *Soit* $k = 1024$*. Expliquez pourquoi la probabilité que le nombre renvoyé ne soit pas de taille 1024 est très petite.*
3. *Expliquez pourquoi cet algorithme ne peut pas être utilisé pour générer des clés RSA (on pourra proposer une méthode efficace qui permette de factoriser un nombre* n *renvoyé par l'algorithme).*

Exercice 82. *Soit* a, b, c *trois entiers et* ExpMod *l'algorithme suivant :*

ExpMod(a, b, c)
$r = 1$
pour $i = 1$ *à* b
  $r = r \times a \mod c$
retourner r

1. *Que calcule* ExpMod *?*
2. *Quelle est la complexité de cet algorithme ?*
3. *Evaluer le temps d'execution si* a, b, c *sont des entiers de 1024 bits.*
4. *Proposer un algorithme rapide calculant la même fonction que* ExpMod*.*

Exercice 83. *Montrer que s'il existe un algorithme rapide pour calculer* $\phi(n)$ *alors il existe un algorithme rapide pour factoriser* n*. Quelles conséquences pour la sécurité de RSA ?*

Exercice 84. *Soit* $n = pq$ *le produit de 2 grands entiers. Supposons que l'on dispose d'un algorithme* ExtractRacine *qui prend en entree* $y \in \mathbb{Z}_n$ *et qui renvoie* $x \in \mathbb{Z}_n$ *tel que* $x^2 \equiv y \mod n$ *si un tel* x *existe.*

1. *Soit* $y \in \mathbb{Z}_n$*. Quel est le nombre possible de racines carrées de* y*, i.e. une racine carrée étant un nombre* $x \in \mathbb{Z}_n$ *tel que* $x^2 = y$*.*

2. *Montrer (ou constater sur des exemples) que pour tout $y \in \mathbb{Z}_n$, la difference de 2 de ses racines carrées (si y en possède) est un multiple de p.*
3. *En déduire un algorithme de factorization de n (qui est efficace si* ExtractRacine *est efficace).*

8.3 Quelques exercices basiques sur RSA/Paillier...

Exercice 85. *Soit $pk = (n = pq, e)$ une clé publique RSA. Peut-on encrypter p avec pk ?*

Exercice 86. *Prouver que RSA est correct ? Autrement dit prouver que pour tout $x \in \mathbb{Z}_n^*$*

$$(x^e)^d \equiv x \mod n$$

Exercice 87. *Soit $n \in \mathbb{N}$. On note m l'inverse de n dans $\mathbb{Z}/\phi(n)\mathbb{Z}$ (on supposera n inversible). Soit $x, r \in \mathbb{Z}/n\mathbb{Z}$ et $X = (1 + x \cdot n)r^n \mod n^2$.*

1. *Montrer que :*
   — $X \equiv r^n \mod n$
   — $r \equiv X^m \mod n$
   — $x = (X \cdot r^{-n} \mod n^2 - 1)/n$
2. *En déduire un cryptosystème.*
3. *Montrer que ce cryptosystème est homomorphique additif, i.e. $D_{sk}(E_{pk}(x).E_{pk}(y)) = x + y$*
4. *Proposez une application à cette propriété d'homomorphie.*

Exercice 88. *Alice a généré une clé publique et secrète du cryptosystème RSA. Bob souhaite lui envoyer un message secret. Pour ceci, il decide de lui envoyer une encryption du code ascii de chaque lettre. Critiquer cette manière de procéder.*

Exercice 89. *Quel est l'étape la plus longue dans la génération de clés RSA ?*

Exercice 90. *Nous souhaitons élaborer un mécanisme de signature électronique permettant de garantir l'intégrité d'un document électronique et d'en authentifier l'auteur, par analogie avec la signature manuscrite d'un document papier. Enoncer des propriétés que devrait vérifier ce mécanisme et proposer une solution.*

Exercice 91. *Alice a généré une clé publique et secrète du cryptosystème RSA. Elle a, en outre, publié une liste de question binaires (dont la réponse est oui/non). Bob souhaite y répondre de manière sécurisé. Proposer un protocole pour réaliser ceci.*

Exercice 92. *Générer une clé publique et secrète du cryptosystème Paillier et les tester. On vérifiera en outre la propriété d'homomorphie. Dire comment ce cryptosystème pourrait être utilisé pour le vote électronique.*

8.4 Quelques attaques...

Exercice 93. *Pour des besoins cryptographiques non explicités ici, Alice doit générer un grand nombre T de clés RSA différentes, i.e. $(pk_1 = \{n_1, e_1\}, sk_1 = \{d_1\})$,...,$(pk_T = \{n_T, e_T\}, sk_T = \{d_T\})$. Afin de gagner du temps, elle décide de ne générer que $t = O(\sqrt{T})$ nombres premiers $p_1, ..., p_t$. On considère l'ensemble N défini par $N = \{p_i p_j \mid (i,j) = \{1, ..., t\}^2, i \neq j\}$*

1. *Montrer que si $t = 2\sqrt{T}$ alors le cardinal de N est supérieur à T.*
2. *Posons $t = 2\sqrt{T}$. Alice choisit arbitrairement T éléments différents $n_1, ..., n_T$ dans N et génère les clés $(pk_1 = \{n_1, e_1\}, sk_1 = \{d_1\})$,...,$(pk_T = \{n_T, e_T\}, sk_T = \{d_T\})$*
   (a) *Comment Alice choisit-elle les valeurs $(e_i, d_i)_{i=1,...,T}$ pour que les clés soient correctes.*
   (b) *Dire pourquoi cette facon de générer les clés s'avère avantageuse en termes de temps de calcul. En vous servant de l'expérience acquise en TP et éventuellement d'arguments plus formels, évaluer le gain de cette méthode par rapport la méthode (classique) consistant à exécuter T fois la fonction* KeyGen*.*
   (c) *Montrer que cette méthode n'est pas judicieuse en terme de sécurité (pensez à Euclide...).*

Exercice 94. *Une voyante australienne réputée prétend avoir deviné les 6 numeros gagnants (compris entre 1 et 49) du prochain tirage du loto. Pour des raisons que nous n'expliciterons pas ici, elle souhaite vous en faire profiter (et seulement vous). Etant d'un naturel peu partageur, vous souhaitez sécuriser la communication. Pour ceci, vous décidez d'utiliser le cryptosysteme RSA en transmettant à cette voyante une clé publique (n, e) de taille 1024.*

1. *Sans consigne de votre part, mais quand même familiarisée avec RSA et la classe* BigInteger*, la voyante vous envoie une encryption de chacun des 6 numeros. Expliquez pourquoi cette façon de procéder n'est absolument pas sécurisée.*
2. *Afin de pallier la faille précedente, vous suggérez à la voyante de vous envoyer une encryption de la concaténation de ces 6 numeros (donnant un nombre à au plus 12 chiffres). En considérant qu'un attaquant dispose de ressources lui permettant d'effectuer une exponentiation modulaire $x^e \mod n$ en 100ms pour tout $x \in \mathbb{Z}_n$, estimez le temps nécéssaire (en moyenne) à l'attaquant pour retrouver les 6 numéros.*
3. *Dans l'euphorie, vous avez malencontreusement choisi $e = 3$. Pourquoi un tel choix n'est-il pas judicieux ? Proposer une attaque quasi-instantanée permettant à l'attaquant de retrouver les 6 numéros (on utilisera le fait que le message encrypté est trop petit par rapport à n).*

Exercice 95. *On suppose que Bob chiffre successivement avec RSA pour Alice deux messages de la forme m et $m + \delta$. Les messages chiffrés correspondants sont c_1 et c_2 (comme dans le fichier qui contient les challenges).*

1. *Combien vaut :*
$$\frac{(m+1)^3 + 2m^3 - 1}{(m+1)^3 - m^3 + 2} \mod N?$$

2. *Sachant que l'exposant public d'Alice est $e = 3$ et que $\delta = 1$, utiliser la relation précédente pour retrouver le message m*

3. *Implémenter cette attaque.*

Exercice 96. *L'échange de clés de Diffie-Hellman fonctionne de la façon suivante : Alice et Bob se mettent d'accord sur un groupe G d'ordre premier p, et choisissent un générateur g (tous les éléments de ce groupe sont donc de la forme g^x, avec x un entier compris entre 1 et $p-1$). De son côté, Alice choisit un entier a compris entre 1 et $p-1$, et calcule g^a. Bob, quant à lui, choisit un entier b compris entre 1 et $p-1$, et calcule g^b. Alice et Bob s'échangent les éléments g^a et g^b, et conservent secrets a et b. Ils peuvent ainsi calculer tous les deux la clé $K = g^{ab}$.*

1. *A quoi peut servir un tel protocole d'échange de clés ?*
2. *Rappelez comment Alice et Bob calculent chacun la clé K.*
3. *A quelle condition algorithmique ce protocole est-il sûr ?*
4. *Plus pratiquement, Alice et Bob utilisent des clés de la forme $g^x \pmod p$ avec $g = 2$ et $p = 29$.*
5. *La clé publique d'Alice est 15. Vous jouez le rôle de Bob : choisissez une clé secrète, calculez la clé publique correspondante, et donnez la clé secrète partagée entre Alice et vous.*
6. *Proposez une généralisation de l'échange de clés Diffe-Hellman à 3 personnes.*
7. *Considérons maintenant l'échange de clés suivant :*
   (a) *Alice tire deux suites binaires aléatoires de longueur n : k et $r \in \{0,1\}^n$. Elle envoie à Bob $s = k \oplus r$.*
   (b) *Bob tire aléatoirement une suite binaire $t \in \{0,1\}^n$ et envoie $u = s \oplus t$ à Alice.*
   (c) *Alice calcule $w = u \oplus r$ et renvoie w à Bob.*
   (d) *Alice choisit la clé k comme clé partagée.*

   Montrez que Bob peut également calculer la clé k à partir des informations reçues. Montrez cependant que ce protocole d'échange de clés n'est pas sûr.

Exercice 97.
1. *Rappelez le fonctionnement du chiffrement RSA.*
2. *Expliquer pourquoi il est dangereux d'attribuer deux couples clé publique/clé secrète (e_1, d_1) et (e_2, d_2) associé chacun à un même module n à deux utilisateurs différents (un chiffré pour l'un peut-il être lu par l'autre ?).*
3. *Supposons que ces deux utilisateurs se fassent mutuellement confiance et qu'un même message m soit envoyé à ces deux utilisateurs : vous allez montrer qu'un espion qui voit passer les deux chiffrés pour ces deux utilisateurs peut retrouver*

le message m sans les clés secrètes. Pour vous aider, considérons l'exemple suivant. Soient $n = 91$, $(e_1, d_1) = (11, 59)$ et $(e_2, d_2) = (5, 29)$ les paires de clés publique/privée de ces deux utilisateurs. Soient $u = 1$ et $v = -2$: combien vaut $e_1 \times u + e_2 \times v$? Calculez les chiffrés c_1 et c_2 du message $m = 2$, et calculez $c_1^u \times c_3^v (\mod n)$. En vous inspirant de ces calculs, donnez une description formelle de l'attaque de RSA dans ce cas pathologique, et montrez qu'elle peut se réaliser en temps polynomial avec les seules données publiques (le pgcd de e_1 et e_2) est à prendre en compte).

8.5 Quelques protocoles...

Exercice 98. *Imaginons qu'Alice (resp. Bob) possède un vecteur secret (à coordonnées entières supposées plus petites qu'une certaine valeur A) $\vec{u}$ (resp. $\vec{v}$). Etablir un protocole* ProdScal *permettant de calculer le produit scalaire $\vec{u}\vec{v}$ sans révéler aucune information sur $\vec{u}$ et $\vec{v}$ (autre que $\vec{u}\vec{v}$). Analyser la sécurité de ce protocole.*

Exercice 99. *Alice possède une 3-DNF secrète f construite à partir de T variables booléennes. Bob possède T variables booléennes secrètes $x_1, ..., x_T$. Construire un protocole* EvalDNF *permettant à Bob d'obtenir $f(x_1, ..., x_T)$ sans rien révéler sur $x_1, ..., x_T$ et sans rien apprendre sur f. Analyser la sécurité de ce protocole.*

Exercice 100. *Alice propose une liste de 10 questions a Bob et n'accepte de répondre qu'a une seule d'entre elle. Bob, quant à lui, souhaite que sa question reste secrète. Autrement dit, Bob choisit secrètement une des 10 questions et souhaite obtenir une réponse sans qu'Alice ne sache quelle question a été choisie et Alice veut avoir la garantie que Bob n'a obtenue qu'une seule réponse. Etablir, si possible, un protocole réalisant ceci.*

Exercice 101. *Proposer une solution pour la distribution de carte virtuelle + analyser + tout autre amélioration ou extension. Concrètement, 4 joueurs en ligne souhaitent se distribuer 8 cartes (pour jouer à la belote par exemple). Chaque joueur devra être assurer qu'aucun autre joueur (ou groupe de joueurs) ne peut tricher, i.e. influencer la distribution, connaitre ses cartes, etc...*

Exercice 102. *Considérons le cryptosystème de Paillier, à savoir les fonctions* Paillier.KeyGen, Paillier.Encrypt *et* Pailler.Decrypt. *Pour simplifier les notations, $[x]_{pk}$ (ou simplement $[x]$ lorsqu'il n'y aura aucune ambiguité sur la clé publique) désignera une encryption d'une valeur x avec la clé publique pk.*

Nous supposerons qu'Alice a généré un couple de clés $(pk, sk) = (n, \phi(n))$ avec la fonction Paillier.KeyGen. *Nous supposerons en outre que Bob dispose de deux encryptions X et Y de deux valeurs[1] x et y i.e. $X = [x]$ et $Y = [y]$. On supposera que x et y sont inconnues de Bob et d'Alice (on pourra supposer qu'elles proviennent d'un protocole antérieur ou d'une tierce partie). L'objet de ce CC est d'établir un protocole entre Bob*

1. Autrement dit, Paillier.Decrypt$(sk, X) = x$ et Paillier.Decrypt$(sk, Y) = y$.

et Alice permettant à Bob d'obtenir une encryption du produit[2] $xy \mod n$ *tout en garantissant que les valeurs* x, y *ne soient dévoilées ni à Bob ni à Alice. Un tel protocole, appelé* Multiplication*, vous est fourni ci-dessous dans une version semi-detaillée.*

Multiplication

Pré-requis : *Bob possède deux encryptions* $[x]$ *et* $[y]$ *de deux valeurs* x *et* y *supposées n'être connues ni de Bob ni d'Alice.*

1. ***Bob*** *envoie des encryptions de* $r+x$ *et de* $s+y$ *où* r *et* s *sont choisis aléatoirement (par Bob) dans* $\mathbb{Z}/n\mathbb{Z}$
2. ***Alice*** *génère et envoie une encryption de* $(x+r)(y+s)$.
3. ***Bob*** *génére et retourne une encryption de* $(x+r)(y+s) - sx - ry - rs$.

Rappel des propriètés homomorphiques du cryptosystème de Paillier. *Soient* $u, v \in \mathbb{Z}/n\mathbb{Z}$, Paillier.Decrypt$([u][v] \mod n^2) = u + v \mod n$ *et* Paillier.Decrypt$([u]^v \mod n^2) = uv \mod n$.

Q1 *A quels ensembles appartiennent les messages et les encryptions considérés par le cryptosystème de Paillier ?*

Q2 *Dire par qui et comment sont utilisées les propriétés homomorphiques additives de Paillier dans ce protocole.*

Q3 *Ecrire le protocole* Multiplication *de manière la plus détaillée possible (en utilisant les fonctions* Paillier.KeyGen, Paillier.Encrypt *et* Paillier.Decrypt*).*

Q4 *Pourquoi Alice n'apprend-elle rien sur* x, y *si Bob respecte le protocole ?*

Q5 *Pourquoi Bob n'apprend-il rien sur* x, y *si Alice respecte le protocole ?*

Q6 *En supposant que Bob et Alice respectent le protocole, justifier le fait que l'encryption retournée par Bob encrypte* $xy \mod n$.

Q7 *Expliquer comment Alice peut agir (en ne respectant pas le protocole) pour que l'encryption retournée par Bob n'encrypte pas le produit* $xy \mod n$.

Q8 *Pourquoi ce protocole ne peut-il pas s'implémenter avec El Gamal (dans sa version homomorphique additif) ?*

8.6 Se tester...

On génère une paire de clés RSA $pk = (n, e), sk = d) \leftarrow RSA.KeyGen(\lambda)$ de **4096** bits. Parmi les affirmations suivantes, dire (en justifiant précisément votre réponse) celles qui sont vraies et celles qui sont fausses.

2. Les propriètés homomorphiques seules de Paillier ne permettent pas à Bob de le faire en local.

1. e peut être choisi égal à 6.
2. Tous les éléments non-nuls de $\mathbb{Z}/n\mathbb{Z}$ sont inversibles.
3. La proportion d'éléments non-inversibles de $\mathbb{Z}/n\mathbb{Z}$ est inférieure à 1%.
4. p' est premier.
5. Il existe un algorithme rapide (de complexité polynomiale) qui prend en entrée n et qui retourne un élément $x \in (\mathbb{Z}/n\mathbb{Z}) \setminus \{0\}$ non inversible.
6. Soit a, b, c trois entiers et considérons l'algorithme A suivant :

   $\mathsf{A}(a,b,c)$
   $R = 1$
   Pour i $= 1$ à b faire
   $\quad R = R \times a \pmod{c}$
   Retourner R

   L'algorithme A peut être utilisé dans la fonction de déchiffrement $RSA.Decrypt$.
7. Quand on chiffre avec $Paillier.Encrypt$ deux fois le même message m avec la même clé publique, les deux chiffrés sont égaux.
8. Si $c = RSA.Encrypt(pk, m)$ et $c' = RSA.Encrypt(pk, m')$. L'élément $c \times c' \pmod{n}$ est un chiffré de $m \times m' \pmod{n}$.
9. Soit r un entier dans $\{1, \ldots, p' - 1\}$. On connait un algorithme rapide qui prend en entrée $g^r \pmod{p'}$ et qui renvoie r.
10. Si la seule méthode de factorisation existante était la méthode des divisions successives, la taille d'un module RSA pour atteindre un niveau de sécurité de 80 bits serait de 160 bits.
11. $33^{33} \pmod 7 \equiv 5^3 \pmod 7$.
12. 12 n'est pas inversible dans $\mathbb{Z}/39\mathbb{Z}$.
13. Soient a, b deux entiers positifs de k bits. Il existe un algorithme rapide (polynomial en k) pow tel que $\mathsf{pow}(a, b) = a^b$.
14. Soient a, b, c trois entiers positifs de k bits. Il existe un algorithme rapide modPow tel que $\mathsf{modPow}(a, b, c) = a^b \mod c$.
15. Soit $x \in (\mathbb{Z}/n\mathbb{Z})^*$. L'inverse modulaire de x est égal à $y = x^{\varphi(n)-1} \mod n$.
16. Si e est un entier de 32 bits alors d est un entier de plus 32 bits.

TP 1

L'objectif de se TP est d'expérimenter quelques notions d'arithmétique modulaire et de découvrir la classe BigInteger *de Java. Tous les entiers manipulés dans ce TP seront des objets de cette classe.*

1. Compter le nombre d'éléments inversibles dans $\mathbb{Z}_{377}$. Le resultat vous surprend-il ?
2. Implémenter un test de primalité naif.
3. Implémenter le test de fermat
4. Combien d'erreurs commet le test de Fermat sur les 10000 premiers entiers ¿2
5. A partir de quelle taille d'entree, votre test naif devient plus lent que le test de Fermat ?
6. Générer 2 entiers premiers p et q de 512 bits.
7. Calculer n=pq et $\phi_n = (p-1)(q-1)$.
8. Générer aléatoirement un nombre e de 16 bits qui soit premier avec ϕ_n.
9. Calculer l'inverse modulaire d de e modulo ϕ_n. Quelle relation vérifient e et d ? La vérifier.
10. Pour plusieurs éléments $x \in \mathbb{Z}_n$, calculer $X = x^e \mod n$ et $X^d \mod n$. Que constate-t-on ?
11. Déduire de ce qui précède un cryptosystème à clé publique. On explicitera précisément les fonctions KeyGen, Encrypt et Decrypt. Quelle est la complexité de ces fonctions ?
12. Dire sur quoi pourrait reposer la sécurité de ce cryptosystème ?
13. On imagine le protocole suivant. Alice génère un couple de clés privée/publique et publie la clé publique avec el cryptosystème précédent. Alice pose des questions binaires (réponse par oui ou par non) à Bob qui lui répond en envoyant une encryption de 1 si la réponse est oui et une encryption de 2 sinon. Dire en quoi ce protocole n'est pas sure. Proposer amélioration (en terme de sécurité) de ce protocole.

TP 2

Vous avez implémenté le cryptosystème de Paillier, à savoir les fonctions Paillier.KeyGen, Paillier.Encrypt et Pailler.Decrypt. On aura aussi besoin de la fonction Paillier.DecryptPlus (légère modification de Paillier.Decrypt) définie par :

Paillier.DecryptPlus prend en entrée une encryption $X = (1 + xn)r^n \mod n^2$ et retourne (x, r).

Pour simplifier les notations, $[x]_{pk}$ (ou simplement $[x]$ lorsqu'il n'y aura aucune ambiguité sur la clé publique) désignera une encryption d'une valeur x avec la clé publique pk.

Nous supposerons dans tout le TP qu'Alice a généré un couple de clés $(pk, sk) = (n = pq, \phi(n))$ avec la fonction Paillier.KeyGen. Nous supposerons en outre que Bob dispose de deux encryptions X et Y de deux valeurs[3] x et y i.e. $X = [x]$ et $Y = [y]$. On supposera que x et y sont inconnues de Bob et d'Alice (on pourra supposer qu'elles proviennent d'un protocole antérieur ou d'une tierce partie). L'objet de ce TP est d'établir un protocole entre Bob et Alice permettant à Bob d'obtenir une encryption du produit[4] $xy \mod n$ tout en garantissant que les valeurs x, y ne soient dévoilées ni à Bob ni à Alice. Un tel protocole, appelé Multiplication, vous est fourni ci-dessous dans une version semi-detaillée.

Multiplication

Pré-requis : *Bob possède deux encryptions X et Y de deux valeurs inconnues x et y.*

1. **Bob** envoie des encryptions de $r+x$ et de $s+y$ où r et s sont choisis aléatoirement (par Bob) dans $\mathbb{Z}/n\mathbb{Z}$
2. **Alice** génère et envoie une encryption de $(x + r)(y + s)$.
3. **Bob** génére et retourne une encryption de $(x + r)(y + s) - sx - ry - rs$.

Q1 Implémenter le protocole Multiplication. Pour cela, on créera 3 méthodes mult1, mult2 et mult3 correspondant respectivement aux étapes 1,2 et 3 du protocole. multk prendra en entrée que ce qui est connu par la partie exécutante (Alice ou Bob) au début de l'étape k. Pour simplifier l'implémentation, "*envoyer à*" sera juste remplacé par "*retourner*". Par exemple mult1 pourra prendre en entrée pk, X, Y et retournera une encryption de $x + r$ et une de $y + s$.

Si Alice dévie du protocole à l'étape 2 en n'envoyant pas une encryption de $(x+r)(y+s)$ alors Bob retournera une encryption incorrecte à l'étape 3. Nous proposons de construire

3. Autrement dit, Paillier.Decrypt$(sk, X) = x$ et Paillier.Decrypt$(sk, Y) = y$.
4. Les propriètés homomorphiques seules de Paillier ne permettent pas à Bob de le faire en local.

un protocole MultiProof qui va permettre à Bob de vérifier qu'Alice a bien respecté l'étape 2.

Plus généralement, supposons qu'Alice (qui possède la clé secrète) dispose de 3 encryptions $[\alpha]$, $[\beta]$ et $[\gamma]$ tel que $\gamma = \alpha\beta \mod n$. Elle souhaite prouver à Bob (appelé le vérifieur) que $\gamma = \alpha\beta \mod n$ sans révéler quoique que ce soit sur ces valeurs à Bob. A l'issue du protocole, Bob devra être certain que cette relation est vérifiée (si elle ne l'est pas, le protocole devra échouer). Nous vous proposons le protocole suivant :

MultiProof

Pré-requis : *Alice propose 3 encryptions* $[\alpha]$, $[\beta]$ *et* $[\gamma]$ *tel que* $\gamma = \alpha\beta \mod n$. *Ces 3 encryptions sont évidemment connues de Bob. Alice souhaite prouver que* $\gamma = \alpha\beta \mod n$ *sans rien devoiler sur ces valeurs. La preuve sera acceptée si le protocole suivant n'échoue pas.*

1. **Alice** choisit δ aléaoirement dans $\mathbb{Z}/n\mathbb{Z}$ et envoie les encryptions $[\delta]$ et $[\pi]$ à Bob où $\pi = \delta\beta \mod n$.
2. **Bob** choisit e aléatoirement dans $\mathbb{Z}/n\mathbb{Z}$ et l'envoie à Alice.
3. **Alice** calcule :
   — $(a, r) \leftarrow$ Paillier.Decryptplus$([\alpha]^e[\delta] \mod n^2)$
   — $(a', r') \leftarrow$ Paillier.Decryptplus$([\beta]^a[\pi]^{-1}[\gamma]^{-e} \mod n^2)$
   et envoie (a, r) et (a', r') à Bob.
4. **Bob** vérifie que :
   (a) $(1 + an)r^n \equiv [\alpha]^e[\delta] \mod n^2$
   (b) $(1 + a'n)r'^n \equiv [\beta]^a[\pi]^{-1}[\gamma]^{-e} \mod n^2$
   (c) $a' = 0$
   Si au moins l'une des trois vérifications échouent alors le protocole échoue.

Q2 Montrer que $a = \alpha e + \delta$ et $a' = 0$ si Alice respecte le protocole.

Q3 Déduire de la question précédente que si Alice respecte le protocole alors Bob n'apprend rien sur α, β, γ (on admettra que r, r' sont independants de α, β, γ et que Paillier (son cryptosystème) est sémantiquement sûr). On utilisera le fait que δ est choisi indépendamment de α.

Q4 Pourquoi les vérifications 4.a et 4.b permettent à Bob de vérifier que $[\alpha]^e[\delta] \mod n^2$ encrypte a et que $[\beta]^a[\pi]^{-1}[\gamma]^{-e} \mod n^2$ encrypte a' ?

Q5 Montrer que si $\gamma \neq \alpha\beta \mod n$ et si les vérifications 4.a et 4.b n'échouent pas alors $a' = 0$ avec une probabilité négligeable (même si $\pi \neq \delta\beta \mod n$). Vous pouvez essayer de montrer qu'elle est inférieure à $\max(1/p, 1/q)$ en utilisant le fait que e est choisi aléatoirement dans $\mathbb{Z}/n\mathbb{Z}$ et que $a' = (\alpha e + \delta)\beta - \pi - e\gamma \mod n$ (voir exercice 1.16).

Q6 Déduire de la question précédente que si $\gamma \neq \alpha\beta \mod n$ alors le protocole échoue avec une probabilité proche de 1.

Q7 Comment MultiProof peut-il être utilisé dans Multiplication pour sécuriser l'étape 2 ?

Q8 Implémenter MultiProof. On créera 4 méthodes MultiP1, MultiP2, MultiP3, MultiP4 en adoptant les mêmes conventions que celles utilisées dans Q1.