

RSA

Travail pratique de mathématiques

Groupe 10

Nicolas Albanesi

Darius Briquet

Abivarman Kandiah

Jonas Stirnemann

11/01/2022

Version: 1.0

Contents

1	Objectif du Travail	2
2	Message reçu	2
3	BLOC	3
3.1	Génération de la paire de clé publique, privé	3
3.2	Exponentiation rapide	4
3.3	Craquage de la clé privée	5
3.4	Algorithme d'Euclide étendu	5
3.5	Inverse multiplicatif	5
3.6	Déchiffage RSA	6
4	Conclusion	7
5	Bibliographie	7

1

Objectif du Travail

Dans ce travail, nous décryptons un message chiffré par l'algorithme RSA. Nous avons reçu un message ainsi que la clé publique utilisée pour le chiffrer, nous devons alors craquer la clé privée à partir de la clé publique pour finalement retrouver le message initial. Pour ce faire, nous avons utilisé différents outils mathématiques formant la base de l'algorithme RSA, nous travaillons dans un domaine d'arithmétique modulaire. Ces outils seront décrits plus en profondeur dans la suite de ce rapport.

2

Message reçu**GROUPE 10**

Langage utilisé : **Python**

Les données du problème:

Partie 1 de la clé publique 'n' : **124344401**

Partie 2 de la clé publique 'e' : **1919**

Le message chiffré : 68393426, 89432542, 115521861, 49234219, 3372048, 122157013, 22349186, 35164123, 34100658, 53001833, 22308360, 27268540, 105477840, 16762760, 113133501, 37039778, 89387590, 82947194, 52601869, 118123698, 68691007, 51830717, 118914410, 22002115, 99813173, 27268540, 120352256, 64650097, 92433385, 414607, 72100751, 23998007, 66133841, 35164123, 74691367, 80836310, 5505466, 59202893, 115463881, 46024313, 74620807, 37361822, 110605220, 80046495, 99953354, 84193611, 89956350, 55477753, 18387435, 5325873, 73193855, 48738452, 33336672, 89963322, 122314690, 4452285, 110289672, 106228774, 31142434, 414607, 88872943, 29217724, 88230256, 89956350, 55477753, 18387435, 108347427, 7851834, 76320201, 34059930, 82599280, 108347427, 92752873, 64650097, 92433385, 414607, 72100751, 112081917, 4440942, 54795823, 120183171, 44819012, 55032737, 87655596, 105198146, 63239385, 33230638, 10365063, 115372183, 71955667, 115564528, 17503687, 115564528, 5325873, 36896144, 99693494, 111498758, 7774636, 105885047,

3

BLOC

3.1

Génération de la paire de clé publique, privé

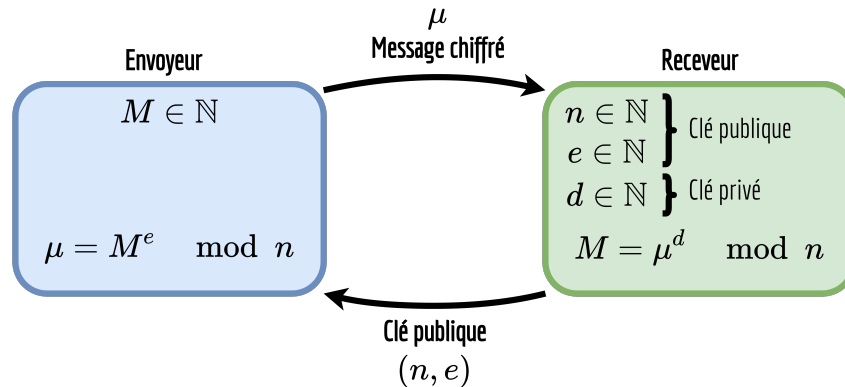


Figure 1: Description globale du RSA

Afin de mieux comprendre les implications du Travail, voici une brève introduction concernant le RSA dans sa généralité. L'algorithme permet de chiffrer un message afin de le transmettre de manière sûr. Pour cela, la personne qui va recevoir le message, crée une paire de clés publique, privé. Puis il envoie cette clé publique à l'envoyeur, qui va alors pouvoir chiffrer ses données avec la clé publique. Le message chiffré est alors envoyé au receveur qui va pouvoir déchiffrer le message. Le principe est que seule la clé privée permet de déchiffrer le message.

Génération de \mathbb{N}

Une partie de la clé publique n est générée par le produit de deux nombres premiers, $p, q \in \mathbb{N}$.

$$n = p \cdot q$$

En général, p et q sont choisis pour être très grands, par exemple sur 512 bits, au vu de la densité de nombre premiers à cet ordre de grandeur, il est **relativement** facile d'en trouver. Nous n'avons qu'à générer un nombre aléatoire de la bonne grandeur puis de décrémenter ou effectuer une opération jusqu'à la primalité.

La sécurité du RSA se repose en fait sur la difficulté à retrouver ces facteurs premiers en un temps réaliste. En effet, il n'existe pas vraiment de méthode pour vérifier la primalité d'un nombre rapidement. Il faut en

h e p i a

fait tester la divisibilité par tous les nombres impaires de 3 à $\sqrt{\text{nombre}}$. Il existe également des méthodes probabilistes telles que le test de Miller-Rabin qui permettent également de tester si un nombre est probablement premier, il se peut alors que le résultat soit faux bien que la probabilité soit très faible.

Génération de Z

Afin de générer la deuxième partie de clé publique, il faut générer un nombre intermédiaire, étant généralement identifié en tant nombre d'Euler. Pour se faire il suffit d'appliquer:

$$Z = (p - 1) \cdot (q - 1) \quad (1)$$

Génération de e

Par la suite, le nombre d'Euler va nous permettre de trouver un nombre e tel que:

$$\text{PGCD}(Z, e) = 1 \quad (2)$$

Génération de d (Clé privé)

Ce e va pour finir nous permettre de calculer la clé privé. Il suffit trouver l'inverse multiplicatif de e grace à la méthode d'Euclide Etendu.

$$\text{PGCD}(Z, e) \Rightarrow e \cdot x + 2 \cdot y \quad (3)$$

$$d = x \mod Z \quad (\text{clé privé}) \quad (4)$$

3.2 Exponentiation rapide

L'exponentiation rapide nous permet de rapidement déchiffrer le message reçu. Le récepteur reçoit le message bloc par bloc, puis il déchiffre chaque bloc individuellement grâce à la formule suivante, avec M le bloc déchiffré, μ le bloc chiffré, d la clé privée et n une partie de la clé publique:

$$M_1 = (\mu_1)^d \mod n \quad (5)$$

Dans notre cas, la formule ci-dessous nous permet de déchiffrer le premier bloc :

$$M_1 = 68393426^{29995379} \mod 124344401 \quad (6)$$

Comme nous pouvons le voir, ce calcul est très grand. Le temps de calcul serait trop long pour déchiffrer chaque bloc, c'est pourquoi l'exponentiation rapide est indispensable.

3.3 Craquage de la clé privée

Afin de craquer la clé privée, il va falloir tout d'abord trouver les facteurs p et q à partir de la clé publique n donnée.

$$n = p \cdot q \quad (7)$$

Pour ce faire, nous allons diviser n par chaque nombre impair de 3 à $\frac{n}{2}$ et vérifier si le résultat est entier. Les pairs p et q sont uniques car premiers.

Lorsque les pairs p et q sont trouvées, on peut commencer à calculer la clé privée.

A l'aide des facteurs p et q , on va pouvoir trouver le facteur z (nombre d'Euler).

$$z = (p - 1) \cdot (q - 1) \quad (8)$$

On va ensuite utiliser l'algorithme d'Euclide étendu afin de trouver l'inverse multiplicatif (x) de la clé publique e modulo z .

$$PGCD(e, z) = e \cdot x + z \cdot y \quad (9)$$

Pour finir à l'aide du facteur x trouvé on peut calculer la clé privée (d).

$$d = x \bmod z \quad (10)$$

3.4 Algorithme d'Euclide étendu

L'algorithme d'Euclide étendu nous permet de calculer les coefficients de Bézout (x et y) de deux entiers a et b tel que

$$PGCD(a, b) = a * x + b * y \quad (11)$$

Ces coefficients seront par la suite utilisés pour calculer l'inverse multiplicatif d'une partie de la clé publique (e), afin de trouver la clé privée.

3.5 Inverse multiplicatif

Afin de simplifier et accélérer nos calculs et de travailler avec des nombres d'ordre correct, nous travaillons dans un corps congruent un nombre Premier. Nous avons alors produit une fonction permettant de trouver l'inverse multiplicatif $\bmod p$ d'un nombre a , ce qui veut dire qu'au lieu de diviser un nombre par une approximation en float, il est possible de le multiplier par l'inverse modulaire. Pour trouver cet inverse, il suffit de trouver les coefficients de Bézout de ce nombre a avec le modulo p . Le premier coefficient donné correspond alors à l'inverse multiplicatif du nombre $a \bmod p$.

$$\mathbf{h \ e \ p \ i \ a} \quad a * u \pmod{p} = 1 \quad (12)$$

3.6 Déchiffrage RSA

Voici une liste des étapes afin de déchiffrer le message reçu, qui est encrypté avec l'algorithme du RSA

Les données du problème:

Partie 1 de la clé publique 'n' : **124344401**

Partie 2 de la clé publique 'e' : **1919**

1 - Bruteforce de p et q

La première étape afin de trouver la clé privée est de trouver les deux facteurs p et q . en utilisant une technique de bruteforce (tester toutes les possibilités possibles).

2 - Trouver la clé privée

La seconde étape est la génération de la clé privée. En effet, la clé privée est générée grâce aux deux facteurs p et q que nous venons de trouver. Il nous est donc possible de régénérer la clé privée.

3 - Déchiffrage des blocs

Le message que nous avons reçu est une liste de blocs (car M doit être plus petit que n). Chaque bloc doit être déchiffré en utilisant la clé privée. Une fois chaque bloc déchiffré, nous devons le décoder afin de le visualiser.

4 - Décoder le message reçu

Le message reçu n'est pas encore lisible. En effet, la manière dont il est codé ne nous permet pas de le lire. C'est pourquoi nous avons décodé le message en inversant l'ordre des bytes (little-endian) de chaque bloc. Suite à cela, il nous suffit de convertir les valeurs unicode de chaque byte en caractère ASCII.

5 - Message final

"Ben si, si c'est l'même volume sonore, on dit "équidistant" [...] S'ils sont équidistants en même temps que nous, on peut repérer le dragon par rapport à une certaine distance. Si le dragon s'éloigne, on s'ra équidistant, mais ça s'ra vachement moins précis et... et pas réciproque."

4

Conclusion

Nous avons réussi à décrypter le message reçu, qui est supposé être impossible à décrypter **car la taille des deux nombres premiers p et q est très faible**. Lorsque le RSA est utilisé correctement, les deux nombres premiers devraient être d'une longueur de 512 bits, ce qui devrait générer un nombre n d'une longueur de 1024 bits. Pour un chiffrement RSA 1024 bits, on admet qu'on puisse tester une clé par cycle d'horloge d'un CPU et qu'on aient autant d'ordinateurs que de particules dans l'univers, donc à peu près 10^{80} ordinateurs et ceux-ci tournent dans une échelle de temps Planck : ($tp = 5.391 * 10^{-44}$) la plus haute vitesse physique possible

$$\frac{2^{1024}}{10^{80}} * 5.391E - 44 = 10^{40}[s] \Rightarrow 10^{32} \text{ années} \quad (13)$$

De ce fait avec les technologies actuelles nous ne pourrions pas craquer l'algorithme par la force brute en un temps réaliste. En effet, l'âge de l'univers n'est que de 14^9 années, cela voudrait dire que nous aurions en moyenne besoin de $4.84003 * 10^{21}$ fois l'âge de l'univers pour y parvenir (ou être chanceux).

5

Bibliographie

https://fr.wikipedia.org/wiki/Temps_de_Planck

https://fr.wikipedia.org/wiki/Test_de_primalit%C3%A9_de_Miller-Rabin