Les secrets de RSA

Karim Zayana, Paul Dorbec

Les secrets de RSA

Karim Zayana (Université de Versailles) et Paul Dorbec (Université de Bordeaux - CNRS)

L'algorithme RSA est le système cryptographique à clé publique le plus répandu. Basé sur l'arithmétique modulaire, il est bien connu des élèves de Terminale des séries scientifiques et de leurs professeurs - mais pas toujours correctement contextualisé. Nous le plaçons ici dans son cadre d'utilisation et l'illustrons avec le langage formel Maple.

De la stéganographie à la cryptographie

Quels qu'en soient les motifs (militaires, médicaux, financiers, amoureux), l'homme a toujours voulu protéger certaines informations. Un premier moyen est de les dissimuler en masquant leur présence ou en les faisant passer pour d'autres. Des exemples courants sont les encres sympathiques qui se révèlent à la bougie, le messager dont il faut raser le crâne pour lire le message (inscrit avant que la chevelure ne pousse), ou plus récemment, les images numériques dont on a imperceptiblement modifié les nuances de rouge/vert/bleu pour y enfouir un message binaire, que l'on retrouve en comparant le code de l'image avec l'image originale.

Un second moyen est le cryptage de l'information : il s'agit de coder le texte d'une façon connue uniquement de l'émetteur ou du récepteur, pour le rendre illisible et non plus invisible. Un exemple classique est le chiffre de César, obtenu par simple transposition des lettres (tel le fameux BcG, décalant le B sur le G). Au lieu de transformer une lettre en une autre, on peut aussi permuter les lettres du message selon le principe des anagrammes. Des chiffrements tels DES ou AES (Data/Advanced Encryption Standard) exploitent les deux principes. Ces *systèmes* de chiffrement sont connus de tous, le secret du message n'étant gardé que par une *clé*. C'est d'ailleurs là un élément majeur de la cryptographie moderne. Il est si difficile de garantir la fiabilité d'un système qu'on préfère le savoir analysé par de nombreux mathématiciens plutôt que fondé sur une idée tenue secrète, peu éprouvée, dont le principe aurait fini par filtrer.

Ici, le *système* est la transposition modulaire, sa *clé K* est la règle

Des clés secrètes aux clés publiques

Quand les clés de codage/décodage sont similaires, on parle de système symétrique. Dans les années 1970, l'algorithme DES était utilisé, on lui donnait alors une durée de vie d'une quinzaine d'années. Il a résisté remarquablement longtemps, et son obsolescence est née de la trop petite taille de sa clé, longue de 56 bits. Il a été remplacé dans les années 2000 par AES, à la clé longue de 128 à 256 bits. Ces systèmes sont très puissants et robustes, et les codage/décodage s'exécutent relativement vite sur des machines dédiées, rendant leur usage très facile.

Pour une transposition, si la clé de codage *K* est BcG, celle de décodage K^{-1} est GcB

Nous avons déjà donné deux schémas élémentaires à clés secrètes : transposition des lettres de l'alphabet, entrelacement de celles du texte. Voici un troisième, adapté aux trames binaires (b_n) comme un flux de données son, ou images. Nous connaissons tous les suites récurrentes linéaires. D'habitude, les coefficients de la récurrence sont des nombres réels. Imaginons maintenant qu'ils soient binaires, que la suite (u_n) soit à valeurs dans $\{0,1\}$, et que les calculs soient menés modulo 2. Par exemple :

$$u_n = u_{n-1} + u_{n-2} + u_{n-4}$$

Les coefficients sont 1,1,0,1 (puis des 0). Avec $u_0 = 1$, $u_1 = 0$, $u_2 = 1$, $u_3 = 1$, nous aurions successivement $u_4 = 1 + 1 + 1 = 1$, $u_5 = 0$, $u_6 = 0$, etc. La suite (u_n) engendrée est pseudo-aléatoire. Imprévisible de l'extérieur, quelqu'un averti des états initiaux et des coefficients de rétroaction sait la régénérer à l'identique. Cette connaissance joue le rôle d'une *clé*. L'émetteur embrouille son message (b_n) en l'ajoutant modulo 2 à (u_n). À son tour, le récepteur bien informé ajoute à la volée au message reçu le même masque (u_n). Cela fonctionne car, modulo 2, $(b_n + u_n) + u_n = b_n + (u_n + u_n) = b_n$. En pratique, la mémoire du système (le nombre de coefficients de la boucle) est grande pour démultiplier les états du registre. Bien entendu, émetteur et récepteur doivent rester synchrones, tout décalage faussant le procédé.

Ici, $K=K^{-1}$

Le grand défaut des algorithmes à clés symétriques est le besoin d'une clé secrète, qu'il faut pouvoir échanger à distance, rapidement, et sûrement. En outre, les communicants devront souvent la modifier : chiffrer un gros volume avec la même clé accroît le risque qu'un tiers ne la brise. D'où l'intérêt des algorithmes à clé publique, dont RSA, qui vont notamment aider à crypter avec un haut degré de confidentialité la clé symétrique à partager.

RSA : principe et illustration avec Maple

RSA tient son nom de ses inventeurs, Rivest, Shamir et Adleman, ingénieurs du MIT (Massachussetts Institute of Technology, Boston, États-Unis) qui l'ont proposé au milieu des années 1970. Sa robustesse vient de la complexité à décomposer un grand nombre en facteurs premiers.

Partons d'un entier $n = pq$ produit de deux premiers distincts p et q. Choisissons une paire d'entiers e et d inverses l'un de l'autre modulo $\varphi = (p-1)(q-1)$.

Pour un entier x, $0 \leq x \leq n-1$, on constate que :

$(x^e[n])^d[n] = x = \left(x^d[n]\right)^e[n]$. La preuve repose sur des résultats algébriques vus en Terminale S (Bezout, Gauss) ou à titre d'approfondissement (Fermat) :

On sait que $(x^e[n])^d = x^{ed}[n]$. Or, $ed = 1 + k(p-1)(q-1)$, k étant un certain entier naturel non nul. Donc :

$$(x^e[n])^d = (x[n].x^{k(p-1)(q-1)}[n]) \quad [n]$$

Comme $x^{k(p-1)(q-1)} = (x^{p-1})^{k(q-1)}$, on distingue naturellement deux cas :

- x premier avec p. Dès lors $x^{p-1} = 1[p]$ par le petit théorème de Fermat. Dans ces conditions, $x^{k(p-1)(q-1)} = 1[p]$, puis $(x^e[n])^d = x[p]$
- x multiple de p. Dans ce cas, $(x^e[n])^d$ l'est aussi, et $(x^e[n])^d = x[p]$ $(= 0[p])$.

De même, $(x^e[n])^d = x[q]$ dans tous les cas. Si bien que :

$$(x^e[n])^d = x + k'p = x + k''q$$

D'après le théorème de Gauss, k' est multiple de q. Bref, $(x^e[n])^d = x[n]$. L'entier x ne dépassant pas n, on a exactement : $(x^e[n])^d[n] = x$.

Définissez maintenant comme ci-dessus les nombres p, q, n, φ, e et d. Le couple (e, n) forme ce qu'on appelle votre *clé publique*, que vous pouvez diffuser largement, tandis que le nombre d est votre *clé privée*, que vous gardez secrète.

Quelqu'un souhaite vous écrire - un texte ou une clé symétrique - que vous seul pourrez lire. Il l'agrège (en mettant bout à bout les valeurs ASCII des lettres par exemple) pour en former un entier $x < n$, ou plusieurs, si le mot est trop long. Il calcule $y = x^e \ [n]$ et vous envoie le résultat. Vous élevez vous-même à la puissance d modulo n le nombre y qu'il a transmis, et traduisez le message en clair. Ce système de codage peut aussi servir à vous authentifier en tant qu'émetteur. Si de la même façon vous codez un message x en $x^d \ [n]$ avec votre clé privée, tout le monde pourra décoder le message avec votre clé publique et s'assurer que vous en êtes bien l'émetteur en le comparant avec le message x initial que vous aurez également transmis.

Avantage : RSA ne requiert pas de clé commune et secrète avant d'être fonctionnel. Subsistent deux inconvénients. Le système ne peut coder les messages que par « petits » paquets, de valeur inférieure à n. Les calculs nécessaires au codage/décodage sont très importants, d'autant plus que n est grand. Aussi n'emploie-t-on pas RSA pour des messages longs ou des applications temps réel.

Sécurité du RSA, ordres de grandeur, et exponentiation rapide

L'entier n choisi aujourd'hui s'écrit souvent sur 1024 à 2048 bits. Il est supérieur à 2^{1023}, soit quelques 340 chiffres décimaux. Si l'on pouvait factoriser n qui est public, on pourrait recalculer $\varphi = (p-1)(q-1)$et retrouver d à partir de e. Cependant, on ne connaît pas à ce jour d'algorithme beaucoup plus efficace que de tenter successivement de diviser n par tous les nombres premiers jusqu'à $\sqrt{n}$. Or $\sqrt{n}$ dépasse 2^{511}. Un théorème de Tchébycheff affirme qu'il y a environ $\frac{2^{511}}{ln(2^{511})}$ nombres premiers inférieurs à $\sqrt{n}$, soit presque $2^{508} > 10^{152}$. À raison de un milliard d'opérations par seconde, cela représente 10^{135} milliards d'années de calcul... Avec le meilleur algorithme connu, on atteint en fait les 10^{14} milliards d'années : cela reste démesuré. Une autre méthode de force brute consisterait à partir d'un mot codé $y = x^e$, d'en calculer les puissances successives y, y^2, y^3, y^4 (modulo n), jusqu'à reconnaître le mot initial x obtenu pour l'exposant d. Cependant, d est en pratique aussi de l'ordre de 2^{1023} et la quantité de calculs reste astronomique. Ces chiffres illustrent combien RSA est gourmand en calculs. Heureusement, existent des méthodes d'exponentiation rapides qui accélèrent les choses. Par exemple, pour obtenir une puissance bien déterminée, par exemple y^{21}, on peut d'abord la fractionner en $y.(y^{10})^2$, puis en $y.((y^5)^2)^2$, puis en $y.((y.(y^2)^2)^2)^2$ avant de remonter la pile de calculs. En bout de chaîne, y^{21} est obtenu sans passer par y^{19}, ni y^{18}, mais grâce à 6 multiplications au lieu de 20. Cette astuce est déjà implantée dans Maple, et appelée par un & commercial : `a &^ d` au lieu de `a ^ d`.

Deux utilisations de RSA

Carte bleue. Dans une transaction par carte bancaire, RSA sert à authentifier l'émetteur de la carte. Le GIE carte bancaire fixe les entiers n, e et d de tout le réseau. Ce triplet unique est rarement changé. Les entiers n et e sont rendus publics (aucune publicité ne les entoure cependant), d est gardé secret. À la fabrication de la carte, un numéro personnel (embossé au recto de la carte) vous est attribué. Il joue le rôle de l'entier x. Sa signature $y = x^d [n]$ est calculée en privé. Les deux nombres sont alors gravés sur la puce de la carte. Puis vous recevez et utilisez votre carte. En caisse,

le terminal de payement lit la paire(x, y), calcule y^e, le compare à x. Le cas échéant, le protocole se poursuit (saisie du code PIN, envoi d'un « challenge », contrôle en ligne, etc.). Au mieux pouvez-vous cloner une carte déjà existante en copiant la paire (x, y). Toutefois, vous ne passerez pas forcément les obstacles suivants.

Protocole SSL. Vous utilisez de nouveau RSA à chaque visite d'un site internet sécurisé dont l'URL commence par *https://*, par exemple en consultant vos mails ou comptes bancaires. Deux conditions sont essentielles à cette sécurisation : crypter les données échangées, et contrôler que la communication se noue avec le « vrai » destinataire. L'adresse IP - seul élément qui identifie simplement votre interlocuteur – varie en effet souvent pour un même site, et peut être falsifiée. La solution ? Quand commence la connexion, le site « sensible » communique (en clair) à votre navigateur un certificat. Ce certificat contient la clé publique du site ainsi qu'une signature de cette clé par une autorité supérieure (elle-même garantie grâce à RSA). Votre navigateur vérifie que la signature correspond à une autorité en qui il a confiance et qu'elle signe bien la clé publique du site, sans quoi il vous alerte. Ensuite, votre navigateur génère aléatoirement une clé symétrique de session qu'il transmet au site « sensible » avec la clé publique qu'il vous a donnée. Le site décrypte la clé symétrique avec sa clé privée pour initier la communication secrète avec vous. Faites le test sur Mozilla depuis gmail : Outils→Informations sur la Page → Sécurité → Afficher le certificat → Détails→ Certificat ou Info clé publique du sujet. Vous lirez : RSA, module n de 1024 bits et affiché en hexadécimal, exposant *e* valant 65537.

Pour aller plus loin : on écoutera avec grand intérêt les conférences de Véronique Cortier, directrice de recherches au CNRS. Vidéos en ligne sur les sites du Collège de France et de l'INRIA.

 # Choix de n

```
> p := nextprime(10^10); q := nextprime(p); n := p*q; phi := (p-1)*(q-1);
    p:= 10000000019   q :=10000000033  n :=  100000000520000000627  phi := 100000000500000000576
```

 # Choix de e (public), calcul de d (privé)

```
> d :=nextprime(10^19); igcdex(d, phi, 'e', 'v'); e:=e mod phi;
    d:= 10000000000000000051   e:= 82199951011149359995
```

```
> mssgAlphanumeriqueClair := "IloveU";        L := length(%);
    "IloveU"     L :=  6
```

 # Padding (mise en forme ASCII)

```
> mssgNumeriqueClair := convert(mssgAlphanumeriqueClair, bytes);
    mssgNumeriqueClair := [73, 108, 111, 118, 101, 85]
> mssgNumeriqueClair := sum(mssgNumeriqueClair[k]*10^(3*(k-1)), k = 1 .. L);
    mssgNumeriqueClair :=   85101118111108073
```

 # Cryptage / décryptage

```
> mssgNumeriqueChiffre := mssgNumeriqueClair &^ e  mod  n;
     mssgNumeriqueChiffre := 4053868297883620455
```

```
> mssgNumeriqueDechiffre := mssgNumeriqueChiffre &^ d  mod  n;
     mssgNumeriqueDechiffre :=  85101118111108073
```

```
> convert(mssgNumeriqueDechiffre, base, 1000);   convert(%, bytes);
    [73, 108, 111, 118, 101, 85]      "IloveU"
```

 # Briser RSA

```
> ifactor(n);
          (10000000019) (10000000033)   # instantané
```