

Chiffrement RSA en Perl 6

Journées Perl 2017

Paris, 9 et 10 juin 2017

Laurent Rosenfeld

Génération de clefs RSA en Perl 6

Perl 6 offre tous les outils pour générer des paires de clefs RSA :

- Entiers de précision arbitraire ;
 - Test de primalité efficace ;
 - Arithmétique modulaire.

Génération de clefs RSA en Perl 6

Perl 6 offre tous les outils pour générer des paires de clefs RSA (et les utiliser pour chiffrer un message secret) :

- Entiers de précision arbitraire ;
 - Test de primalité efficace ;
 - Arithmétique modulaire.

Chiffrement RSA (1)

- Inventé par Rivest, Shamir et Adleman, du MIT (1977)
- Chiffrement asymétrique utilisant une paire de clefs (des nombres entiers) :
 - Une clef publique (ou fonction à sens unique) pour chiffrer un texte, et
 - Une clef privée (ou brèche secrète) pour déchiffrer un cryptogramme (un texte préalablement chiffré)
- Ces clefs peuvent aussi servir à signer (authentifier) des messages.

Chiffrement RSA (2)

- Alice désire recevoir des données confidentielles. Elle génère deux clefs :
 - Elle rend la clef publique accessible à ses correspondants
 - Bob utilise cette clef publique pour chiffrer son message destiné à Alice
 - Alice utilise sa clef privée pour le déchiffrer
- Ce mécanisme est souvent utilisé pour échanger une clef symétrique secrète pour encoder le vrai message (PGP, Gnu-PG)

Les clefs

- L'algorithme utilise les congruences sur les entiers et le petit théorème de Fermat pour obtenir des fonctions à sens unique
 - Tous les calculs sont faits modulo un nombre n qui est le produit de deux entiers premiers
 - Les messages (clairs et chiffrés) sont des entiers d'une taille inférieure à n
 - Le chiffrement et le déchiffrement consistent à élever le message à une certaine puissance modulo n .

Génération des clefs

- Le création des clefs se fait comme suit :
 - Choisir deux nombres premiers distincts p et q
 - Calculer leur produit $n = pq$
 - Calculer $\varphi(n) = (p-1)(q-1)$ (fonction d'Euler)
 - Choisir un entier e premier avec $\varphi(n)$ et strictement inférieur à $\varphi(n)$: exposant de chiffrement
 - Calculer l'entier d , inverse de e modulo $\varphi(n)$, l'exposant de déchiffrement (e^{-1})
 - (n, e) : clef publique, (n, d) : clef privée
 - L'algo repose sur la difficulté de factoriser n

Chiffrement et déchiffrement

- Si M est un entier inférieur à n représentant le message, alors le message chiffré C est :
 - $C \equiv M^e \pmod{n}$
 - (C est choisi strictement inférieur à n)
- Pour déchiffrer, on utilise d , l'inverse de e modulo $(p-1)(q-1)$:
 - $M \equiv C^d \pmod{n}$

Exemple (avec des petits nombres)

- Génération des clefs

- On choisit deux nombres premiers, par exemple 5 et 11
- Leur produit $n = 5 \times 11 = 55$ est le module de chiffrement
- Valeur de la génératrice d'Euler : $\varphi(n) = (5-1)(11-1) = 40$
- On choisit l'exposant de chiffrement $e = 3$, premier avec 40
- L'exposant de déchiffrement est 27, l'inverse de e modulo 40
(car $3 \times 27 = 81 \equiv 1 \pmod{40}$)
- La clef publique est (55, 3) et la clef privée (55, 27)

- Chiffrement et déchiffrement

- Chiffrement de $M = 42$: $C = 42^3 = 74\,088 \equiv 3 \pmod{55}$
- Déchiffrement de C : $M = 3^{27} \equiv 42 \pmod{55}$
(car $3^{27} = 7625597484987 = 42 + 55 \times 138647226999$)

Les fonctionnalités de Perl 6

- Entiers de précision arbitraire :

```
say 3**27 - ((55 * 138647226999) + 42); # 0
```

- Test de primalité (de Miller-Rabin)

```
say is-prime 2**521 - 1; # -> True
```

- La fonction *expmod(x, y, z)* renvoie $x^y \bmod z$

```
say expmod(4, 2, 5);      # 1   (équiv 4**2 % 5)
say expmod 3, 27, 55;     # -> 42
```

- gcd* renvoie le PGCD de deux nombres

```
say 15 gcd 45;           # -> 15
```


Procédure de génération des clefs

```
sub premier-aléatoire(Int $digits) {  
  loop {  
    my $try = (10**$digits .. 10**($digits+1)).pick;  
    return $try if $try.is-prime;  
  }  
}  
  
sub génère-clef (Int $nb-digits) {  
  my $p = premier-aléatoire $nb-digits;  
  my $q = premier-aléatoire $nb-digits;  
  my $n = $p * $q;  
  my $φ = ($p-1) * ($q-1);  
  my $e;  
  loop {  
    $e = (1..^$n).pick;  
    last if $e gcd $φ == 1;  
  }  
  my $d = expmod($e, -1, $φ);  
  my $clef-publique = [ $e, $n ];  
  my $clef-privée = [ $d, $n ];  
  return $clef-publique, $clef-privée;  
}
```


Utilisation

- Procédures de chiffrement et déchiffrement :

```
sub chiffre-nb($message, $clé-pub) {  
    return expmod($message, $clé-pub[0], $clé-pub[1])  
}
```

```
sub déchiffre-nb($message, $clé-priv) {  
    return expmod($message, $clé-priv[0], $clé-priv[1])  
}
```

- Exemple d'appel de l'ensemble

```
my ($clef-publique, $clef-privée) = génère-clef 180;  
say $_ for $clef-publique, $clef-privée;  
say déchiffre-nb chiffre-nb(1234567890, $clef-publique),  
    $clef-privée;      # -> 1234567890
```


Conclusion

- Facile, non ? Et fun, non ?
- Fait en quelques lignes de code
 - Remarque : j'ai décrit l'algo de base, je ne dis pas que le code soit cryptographiquement fiable
- Des questions ?

Liens :

- <https://www.promptworks.com/blog/public-keys-in-perl-6>
- <http://shop.oreilly.com/product/0636920065883.do>
- <http://greenteapress.com/wp/think-perl-6/>

Tests

- Utilisation du module Tests :

```
use Tests;
my $début = now;
plan 20;
for 1..20 {
    my $msg = (1..$clef-privée[1]).pick;
    my $crypt = chiffre-nb $msg, $clef-publique;
    my $decrypt = déchiffre-nb $crypt, $clef-privée;
    ok $decrypt == $msg;
}
say "duration: ", now - $début;
```

- Imprime « OK 1 » ... « OK 20 »
- Durée d'exécution : « duration: 0.10658438 »

Chiffrer un message texte

- Pour chiffrer un texte, on peut le convertir en utilisant la représentation UTF-8 de la chaîne

```
sub chiffre-texte ($txt-msg, $clé-pub) {  
  return chiffre-nb $txt-msg.encode.List.fmt("%02x",  
    "").map( { :16($_) } ).join, $clé-pub;  
}  
sub déchiffre-texte ($schiffre, $clé-priv) {  
  my @octets = déchiffre-nb($schiffre,  
    $clé-priv).join.Int.base(16).comb(2).map({ :16($_) });  
  return Blob.new(@octets).decode;  
}  
my $code = chiffre-texte "Maître Corbeau sur un arbre perché",  
  $clef-publique;  
say déchiffre-texte $code, $clef-privée;  
# -> Maître Corbeau sur un arbre perché
```