

Cryptographie – TD3

Jérémy Briffaut

Jean-Christophe Deneuville

<jeremy.briffaut@insa-cvl.fr>

<jean-christophe.deneuville@insa-cvl.fr>

Lundi 1er octobre 2018

Téléchargez et décompressez l'archive TD3.zip.

Rappels de cours RSA [1] :



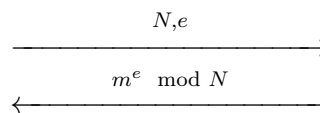
p, q grand premiers, $N = pq$
 e premier avec $\varphi(N) = (p-1)(q-1)$

$$d = e^{-1} \mod \varphi(N)$$

$$(m^e)^d \mod N = m$$



message $m \in \mathbb{Z}/N\mathbb{Z}$



$$m^e \mod N$$

(petit) **Théorème de Fermat** : Si p est premier, alors $\forall a \in \mathbb{Z}/p\mathbb{Z}$, on a : $a^{p-1} = 1 \mod p$.

Conséquence : $\forall m \in \mathbb{Z}/N\mathbb{Z}$, on a : $m^{ed} \mod N = m^{ed \mod \phi(N)} = m^1 = m \mod N$.

Exercice 1 Chiffrement RSA

1. Étudiez, compilez, et tester le code `test_size.c`.
Que fait ce programme ?

On définit un nouveau type d'entier, le `huge` :

```
typedef unsigned long long int huge;
```

2. Pourquoi utiliser le type `huge` plutôt que `int` lors de l'utilisation de RSA ?
3. Quelles sont les valeurs minimales et maximales pouvant être stockées sur un `int` ? Sur un `huge` ?
4. Que fait la fonction suivante ?

```
static huge modexp(huge a, huge b, huge n) {
    huge y;
    y = 1;
    while (b != 0) {
        if (b & 1)
            y = (y * a) % n;
        a = (a * a) % n;
        b = b >> 1;
    }
}
```

```

    }
    return y;
}

```

La génération des clés RSA nécessitant plusieurs routines, elle fera l'objet de l'exercice suivant.

5. À l'aide de la fonction `modexp`, écrire une fonction `rsa_encrypt` qui prend en paramètres une clé publique `(e, N)` et un message `m` à chiffrer (de type `huge`) et qui renvoie le message chiffré avec l'algorithme RSA. Tester votre fonction avec les valeurs vues en cours (slide 28).
6. De la même façon, écrivez une fonction `rsa_decrypt` qui prend en paramètre une clé privée `d` et un message chiffré `c` (de type `huge`) et qui renvoie le message déchiffré correspondant. Tester votre fonction avec les valeurs vues en cours (slide 28).

Exercice 2 Génération des clés RSA

1. Avant de pouvoir générer des clés RSA, nous avons besoin de sous-routines.
 - a) Écrivez une fonction `gcd(a, b)` qui calcule le Plus Grand Commun Diviseur (PGCD) de `a` et `b`. Pour rappel, l'[algorithme d'Euclide](#) permet de faire ceci [2].
 - b) Écrivez une fonction `secret_keyGen(e, phi)` qui, à partir de `e` et `phi` tels que $\text{PGCD}(e, \text{phi}) = 1$, retourne `d` tel que $e \cdot d = 1 \pmod{\text{phi}}$. Pour rappel, l'[algorithme d'Euclide étendu](#) permet de trouver `(u, v)` tels que $e \cdot u + \text{phi} \cdot v = \text{gcd}(e, \text{phi})$ [3].
 - c) Écrivez un générateur aléatoire de nombres premiers (c'est-à-dire une fonction qui renvoie un nombre premier aléatoire entre 0 et une certaine valeur `N` passée en paramètre).
2. Écrivez maintenant une fonction `keyGen` qui génère une paire de clés RSA (`pk`, `sk`) avec `pk = (e, N)` et `sk = d`.

Exercice 3 Librairie de chiffrement

1. Compléter le code de la librairie en complétant les fonctions `rsa_encrypt` et `rsa_decrypt`. Pour cela, vous utiliserez les fonctions `texttoint` et `inttotext` pour normaliser le message et le caractère '\$' comme séparateur de bloc.
2. Inclure votre générateur de clés RSA dans la librairie.

Références

- [1] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2) :120–126, 1978.
- [2] Wikipedia contributors. Algorithme d'euclide — Wikipedia, the free encyclopedia, 2018. [Online; accessed 27-September-2018].
- [3] Wikipedia contributors. Algorithme d'euclide Étendu — Wikipedia, the free encyclopedia, 2018. [Online; accessed 27-September-2018].