ISIFC 3, Crypto
Chiffrement asymétrique

Jean-François COUCHOT
Université de Franche-Comté, UFR-ST

Plan

Principe de la cryptographie asymétrique

RSA : Rivest, Shamir, Adelman

Plan

Principe de la cryptographie asymétrique

RSA : Rivest, Shamir, Adelman

Deux clefs : une publique K_A et une privée K_A^{-1}

Deux clefs pour deux rôles

En *cryptographie asymétrique* ou *cryptographie à clef publique*, Alice a un couple de clefs (K_A, K_A^{-1}) :

- K_A, sa clef publique qu'elle distribue
- K_A^{-1}, sa clef privée qu'elle conserve

Il existe deux utilisations principales de ces clefs :

- Le chiffrement : c'est l'expéditeur qui chiffre avec la clef publique du destinataire
- La signature : c'est le signataire qui encode avec sa clef privée

Chiffrement

Chiffrement

- Bob veut envoyer un message M à Alice
- Il récupère K_A, la clef publique d'Alice
- Il envoie $C = E(M, K_A)$ à Alice
- Alice utilise alors sa clef privée K_A^{-1}
- Elle calcule $D(C, K_A^{-1}) = M$

Propriété obtenue : confidentialité

Signature

Signature

- Alice veut signer un message M pour Bob
- Elle calcule $S = D(H(M), K_A^{-1})$, avec $H(M)$ empreinte de M
- Elle envoie M et S à Bob
- Bob récupère K_A, la clef publique d'Alice
- Il calcule $E(S, K_A)$ et $H(M)$ et vérifie que les résultats sont egaux

Propriété obtenue : authentification

Cryptographie symétrique vs asymétrique

Comparaison

- Cryptographie symétrique
  - Une clef secrète K_{AB}
  - ☺ Chiffrement rapide
  - ☹ Une clef par couple d'acteurs
- Cryptographie asymétrique
  - Une clef publique K_A
  - Une clef privée K_A^{-1}
  - ☺ Un couple de clefs par acteur
  - ☹ Chiffrement lent

→ On utilise le chiffrement asymétrique pour partager une clef de chiffrement symétrique secrète

Plan

Principe de la cryptographie asymétrique

RSA : Rivest, Shamir, Adelman

- Généralités et rappels
- Chiffre RSA en 4 étapes
- Mise en pratique de RSA

Plan

Principe de la cryptographie asymétrique

RSA : Rivest, Shamir, Adelman
- Généralités et rappels
- Chiffre RSA en 4 étapes
- Mise en pratique de RSA

Historique

⋆

Historique de RSA

- 1977 : naissance de l'algorithme [1] par Rivest, Shamir, Adelman
- 1983 : breveté aux USA jusqu'en 2000
- mais 1973 : algorithme équivalent trouvé par Clifford Cocks (British intelligence agency Government Communications Headquarters) mais classifié secret jusqu'en 1997

1. Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.

Fondements théoriques : arithmétique

- il est possible de trouver trois très grands nombres entiers positifs e, d et n tq. pour tout message m, $0 \leq m \leq n$

$$(m^e)^d \equiv m \mod n \tag{1}$$

- et connaissant e et n (et même m), il est très difficile de trouver d

Rappels d'arithmétique, diviseur, pgcd

Division euclidienne, $a \in \mathbb{N}$ et $b \in \mathbb{N}^*$

$\exists\ q \in \mathbb{N}$ (quotient) et $r \in \mathbb{N}$ (reste) uniques, tq. $a = b \times q + r$ et $r < b$

Plus grand diviseur commun pgcd(a, b), $a \in \mathbb{N}^*$ et $b \in \mathbb{N}$

pgcd(a, b) est l'entier naturel qui vérifie :

$$\left.\begin{array}{l} \bullet \mathrm{pgcd}(a, b) | a \text{ et } \mathrm{pgcd}(a, b) | b; \\ \bullet d | a \text{ et } d | b,\ \text{ alors } d | \mathrm{pgcd}(a, b). \end{array}\right\} \text{ en particulier } \mathrm{pgcd}(a, 0) = a$$

Ex. : $550 = 2 \times 5^2 \times 11$ et $1540 = 2^2 \times 5 \times 7 \times 11 \leadsto$
$\mathrm{pgcd}(550, 1540) = 2 \times 5 \times 11 = 110$

Primalité

- p est premier s'il n'a que 2 diviseurs, 1 et p
- a et b sont premiers entre eux si $\mathrm{pgcd}(a, b) = 1$

Euclide, identité de Bézout

Algorithme d'Euclide : $\text{pgcd}(a, b)$, $a \in \mathbb{N}^*$ et $b \in \mathbb{N}$

- Si $a = bq_1 + r_1$, $0 \leq r_1 < b$, alors $\text{pgcd}(a, b) = \text{pgcd}(b, r_1)$
  - A prouver : « d divise a et b » $\Leftrightarrow$ « d divise b et r_1 »
- Si $r_1 = 0$ on a $\text{pgcd}(a, b) = \text{pgcd}(b, 0) = b$, fini
- Sinon, division $b = r_1 q_2 + r_2$, $0 \leq r_2 < r_1$, et $\text{pgcd}(b, r_1) = \text{pgcd}(r_1, r_2) \ldots$
- Ex. $\text{pgcd}(1540, 550)$? $1540 = 550.2 + 440$, $550 = 440.1 + 110$, $440 = 110.4 + 0 \rightsquigarrow \text{pgcd}(1540, 550) = 110$

Identité de Bézout pour a et b dans N^*

Il existe x et y entiers tq. $a.x + b.y = d$, avec $d = \text{pgcd}(a, b)$

- Preuve : appliquer Euclide et remonter

Trouver x et y tq $29x + 72y = 1$

- Calcul de $\text{pgcd}(72, 29) = 1$:

$$\begin{aligned} 72 &= 29 \times 2 + 14 \\ 29 &= 14 \times 2 + 1 \\ 14 &= 1 \times 14 + 0 \end{aligned}$$

- "On remonte" à partir de $1 = \text{pgcd}(72, 29)$

$$\begin{aligned} 1 &= 29 - 14 \times 2 \\ 1 &= 29 - (72 - 29 \times 2) \times 2 \\ 1 &= 29 \times 5 + 72 \times (-2) \end{aligned}$$

Plan

Principe de la cryptographie asymétrique

RSA : Rivest, Shamir, Adelman

- Généralités et rappels
- Chiffre RSA en 4 étapes
- Mise en pratique de RSA

Les 4 étapes du chiffre RSA

1. Alice, destinataire génère de 2 grands nombres premiers
2. Alice choisit sa clef publique K_A, la partage et construit sa clef privée associée K_A^{-1}
3. Bob chiffre M selon $C = E(M, K_A)$ et envoi C à Alice
4. Alice déchiffre C selon $D(C, K_A^{-1})$, c'est M

Alice génère 2 grands nombres premiers p et q

Taille de p et q ?

- Ordre de grandeur de p et q : 1024 bits ($\approx$ 309 chiffres, 64 hex)
- Ordre de grandeur de $n = p \times q$: 2048 bits ($\approx$ 617 chiffres, 128 hex)

Obtenir un grand nombre premier : par générer tester

- En utilisant un PRNG cryptographiquement sûr
- En testant la primalité du nombre engendré (par Miller-Rabin, vu à la fin)

Calcul de la valeur de l'indicatrice d'Euler $\varphi(n)$

- Par définition : $\varphi(n) = (p-1)(q-1)$
- Si l'adversaire connaît n, mais ni p, ni q
  - Peut essayer de factoriser n, mais problème très difficile
  - Ne peut donc pas calculer $\varphi(n)$

Exemple jouet

- $p = 7$, $q = 13$, $n = 91$, $\varphi(91) = 72$

Alice construit ses clefs K_A et K_A^{-1}

La clef publique $K_A = (e, n)$ à diffuser

- Avec e, $1 \leq e < \varphi(n)$, e premier avec $\varphi(n)$
- $K_A = (e, n)$ peut être partagée pour chiffrer des messages

La clef privée $K_A^{-1} = (d, n)$ à conserver

- e et $\varphi(n)$ premiers entre eux :
  - Bézout : $\exists$ d et y tq. $e.d + \varphi(n).y = 1$ $(ed \equiv 1 \mod \varphi(n))$
- $K_A^{-1} = (d, n)$ doit être conservée par Alice pour déchiffrer

Poursuite de l'exemple

- 29 et 72 premiers entre-eux : $K_A = (29, 91)$ est une clef publique
- $1 = 29 \times 5 + 72 \times (-2)$ (T13) : la clef privée associée est $K_A^{-1} = (5, 91)$

Bob chiffre M selon $C = E(M, K_A)$

Découpage en bloc de k bits

- k : plus grand entier tel que $2^k < n$ (n est connu)
- chaque bloc de M : vu comme un entier m ($m \leq 2^k - 1 < n$)

Chiffrement de m avec $K_A = (e, n)$

- $c \equiv m^e \mod n$

Poursuite de l'exemple : chiffrement de $M = 59$ avec $K_A = (29, 91)$

- $k = 6$ ($2^6 = 64 < 91 \leq 2^7 = 128$) $\leadsto$ chiffrement de $M = m = 59$ possible
- $59^{29} \mod 91$: ne pas évaluer d'abord 59^{29} puis la division par 91 ensuite
- Exponentiation rapide : décomposition
  - $59^{29} \mod 91 = 59^1.59^4.59^8.59^{16} \mod 91 = 59.74.16.74$ $\mod 91 = 89 \mod 91$

Alice déchiffre C selon $M = D(C, K_A^{-1})$

Déchiffrement de c avec $K_A^{-1} = (d, n)$

- Chaque mot c : décodé en un mot m' sur k bits par
  - $m' \equiv c^d \mod n$, par exponentiation rapide aussi
- Concaténation des mots m' sur k bits pour obtenir M' (=M ?)

Poursuite de l'exemple : déchiffrement de $c = 89$ avec $K_A^{-1} = (5, 91)$

- $89^5 \mod 91 = 89^1.89^4 \mod 91 = 89.16 \mod 91 = 59 \mod 91$
- $m' = 59 = M' = M$

Plan

Principe de la cryptographie asymétrique

RSA : Rivest, Shamir, Adelman

- Généralités et rappels
- Chiffre RSA en 4 étapes
- Mise en pratique de RSA

Clés : génération et sauvegarde

```python
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

rsa_obj = RSA.generate(2048)
rsa_pub = rsa_obj.publickey()

with open('mPubKey.pem','wb') as f:
    f.write(rsa_pub.exportKey('PEM'))

with open('mPrivKey.pem','wb') as fp:
    fp.write(rsa_obj.exportKey('PEM'))
```

Chiff^t à p. de la clé publique

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP


with open('mPubKey.pem','rb') as f:
    pubKey = RSA.importKey(f.read())
    encryptor = PKCS1_OAEP.new(pubKey)
    message_to_encrypt = 'Securité L3!'
    ciphertext = encryptor.encrypt(message_to_encrypt.encode('utf-8'))

with open("cipher.txt","wb") as fc :
    fc.write(ciphertext)
```

Déchiff^t à p. de la clé privée

```python
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP


with open("cipher.txt","rb") as fd :
    ciphertext = fd.read()

with open('mPrivKey.pem','rb') as fe:
    privKey = RSA.importKey(fe.read())
    decryptor = PKCS1_OAEP.new(privKey)
    decrypted_message = decryptor.decrypt(ciphertext)
    print(decrypted_message.decode('utf-8'))
```