INSA de Lyon
Département Télécommunications,
Services et Usages

4TC-SEC
TD Cryptographie asymétrique

François Lesueur <francois.lesueur@insa-lyon.fr>

Durée : 2h

Ce TD présente et applique les notions de cryptographie asymétrique vues en cours :
- Génération de clés RSA
- Distribution de clés
- Signature et chiffrement RSA

Vous pourrez par exemple utiliser python pour calculer les exponentiations modulaires. Lancez python, puis dans l'interpréteur tapez `pow(a,b,c)` pour obtenir $a^b[c]$. Vous pouvez également utiliser `http://www.wolframalpha.com`.

1 Définition du cryptosystème

Le cryptosystème que nous allons utiliser ici est basé sur la fonction RSA. Le cryptosystème proposé est simple et présente donc certaines vulnérabilités mais illustre le fonctionnement. Cette partie définit le cryptosystème, il n'y a rien à faire ici.

1.1 Génération de clés RSA

Voici l'algorithme simplifié de génération de clés RSA (en réalité, d'autres tests doivent être réalisés) :
- Choisir deux nombres premiers p et q (liste un peu plus loin)
- Calculer $n = p \times q$
- Calculer $\phi(n) = (p-1)(q-1)$
- Choisir e tel que :
  - $1 < e < \phi(n)$
  - $pgcd(e, \phi(n)) = 1$
  - Par exemple, un premier qui ne divise pas $\phi(n)$
- Déterminer $d \equiv e^{-1}\ mod\ \phi(n)$

L'exemple est réalisé avec $p = 31, q = 37, n = 1147, \phi(n) = 1080, e = 7, d = 463$.

La clé publique est (e, n), ici $(7, 1147)$, et la clé privée est (d, n), ici $(463, 1147)$. La propriété utilisée est que pour tout message m, $m^{de}[n] = m$

1.2 Chiffrement et déchiffrement

Nous allons chiffrer des chaînes de caractères. Pour cela, chaque lettre est remplacée par son rang dans l'alphabet, sur 2 chiffres :

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	
01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

Par exemple, "crypto" devient `03 18 25 16 20 15`

Ensuite, afin de ne pas retomber dans un chiffrement par substitution simple, les chiffres sont assemblés par blocs de 3 (complété éventuellement de 0 à la fin), ainsi `03 18 25 16 20 15` devient `031 825 162 015`.

Enfin, chaque bloc clair de 3 chiffres est chiffré indépendamment par la fonction RSA :

$$bloc_{chiffré} = bloc_{clair}{}^{e}[n]$$

Attention, (e, n) représente une clé publique, mais celle de qui ? L'utilisation de la clé $(7, 1147)$ donne le chiffré `1116 751 245 1108`.

Le déchiffrement est opéré de manière réciproque, en utilisant la clé privée au lieu de la clé publique. Chaque bloc clair est réobtenu à partir du bloc chiffré par le calcul : $bloc_{clair} = bloc_{chiffré}{}^{d}[n]$.

1.3 Signature et vérification

Nous allons signer des chaînes de caractères. Pour cela, chaque lettre est remplacée par son rang dans l'alphabet. Pour un message $m = (m_0, \ldots, m_i)$ avec $(m_0, \ldots, m_i)$ les rangs de chaque lettre (attention, on ne fait plus des blocs de 3 chiffres ici), le haché $h(m)$ est calculé par l'algorithme suivant :

$h \leftarrow 2$
for $j = 0..i$ **do**
  $h \leftarrow h \times 2$
  $h \leftarrow h + m_j$
end for
return $h \ mod \ 1000$

La valeur de la signature vaut alors $h(m)^d[n]$. Attention, (d, n) représente une clé privée, mais celle de qui ? Le haché de "crypto" vaut par exemple 151 et la signature par $(463, 1147)$ est 215.

Le message est alors envoyé accompagné de sa signature. La vérification d'un message reçu m signé avec sig est opérée de la manière suivante :

- Calculer $h(m)$ par rapport au m reçu
- Calculer $sig^e[n]$
- Vérifier que $h(m) == sig^e[n]$ sur le message reçu

2 Génération des clés

Nous allons commencer par générer une paire de clés RSA pour chacun. Utilisez pour cela l'algorithme présenté précédemment. Gardez votre clé privée secrète et transmettez votre clé publique avec votre nom à l'enseignant, sur un papier. Elle sera inscrite au tableau (la "PKI").

Une petite liste de nombres premiers pour gagner du temps :

31	37	41	43	47	53	59	61	67	71	73	79	83	89	97	101	103	107
109	113	127	131	137	139	149	151	157	163	167	173	179	181	191	193	197	199
211	223	227	229	233	239	241	251	257	263	269	271	277	281	283	293	307	311
313	317	331	337	347	349	353	359	367	373	379	383	389	397	401	409	419	421

Code Python pour calculer a^{-1} *mod* b : `modinv(a,b)` (disponible sur Moodle, `from modinv import *`) :

```
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m
```

3 Échange de messages chiffrés

Vous allez maintenant transmettre un message chiffré à un étudiant éloigné par un protocole multi-saut : vous le transmettez à un voisin, qui le redonne à un voisin, *etc.*, jusqu'à sa destination. Vous jouerez à la fois les rôles d'émetteur, de routeur (malicieux ou non) et de récepteur. Le chiffrement assure la *confidentialité* du message transmis.

1. **Envoi de votre message** : Chiffrez un message de votre choix avec le cryptosystème proposé. Inscrivez sur un papier votre identité, le message chiffré et le destinataire. Envoyez-le !
2. **Routage des autres messages** : Que fait un routeur ? Il lit un message, l'analyse, décide où l'envoyer puis le reproduit. De manière analogue, vous allez pour chaque saut retransmettre le message entrant mais vous pouvez le lire avant de le retransmettre. Pouvez-vous en déduire des informations ?
3. **Réception d'un message** : À la réception d'un message, appliquez l'algorithme de déchiffrement. Quelqu'un d'autre sur la route du message pouvait-il obtenir le clair de ce message ?

4 Échange de messages signés

Vous allez maintenant transmettre un message clair signé à un étudiant éloigné par ce même protocole multi-saut. La signature permet de vérifier l'*intégrité* du message transmis.

1. **Envoi de votre message** : Signez un message de votre choix avec le cryptosystème proposé. Inscrivez sur un papier votre identité, le message clair, la signature et le destinataire. Envoyez-le !
2. **Routage des autres messages** : Utilisez le même protocole multi-saut que précédemment. Pour chaque saut, recopiez le message entrant sur un autre papier puis retransmettez ce second papier.
3. **Réception d'un message** : À la réception d'un message, appliquez l'algorithme de vérification de la signature. Le message reçu est-il intègre ? Si non, quelle attaque avez-vous détectée ?

5 Attaques sur le cryptosystème proposé

Étudiez et testez quelques attaques sur le système mis en place :
– Modification de message en conservant la validité de la signature
– Attaque de la clé privée (par factorisation de n par exemple)
– Attaque à message choisi
– ...

Toutes ces attaques sont possibles ici. Réfléchissez à leur cause et aux protections mises en place dans les cryptosystèmes réels. Implémentez une (ou plusieurs) attaque dans le langage de votre choix, proposez une contre-mesure et évaluez la complexité rajoutée par votre contre-mesure.