Composition de Cryptographie - 2008

Il est recommandé aux élèves de bien choisir l'ordre des questions selon leurs compétences et rapidités.

Si l'élève n'arrive pas à faire une démonstration, il peut considérer que le résultat de la démonstration est admis sur le reste l'exercice.

Le support de cours et les calculatrices sont permis.

1. SHA1

a) Pour quelle raison une information de taille nulle doit avoir une valeur hash ?
b) A partir du schéma d'implémentation du SHA1 expliquer comment le SHA1 appliqué à une information de taille nulle donne un résultat.

2. Générateurs aléatoires équiprobables

Modifier la routine

```
int range_rand(int mini; int maxi)
{
        return ((int) long_rand() % (maxi – mini+ 1)  + mini);
}
```

pour générer des valeurs aléatoires équiprobables entre mini et maxi.

3. Cartes à Puce RSA

Le but de cet exercice est d'étudier la possibilité pour carte à puce RSA de type basiccard pour réaliser le RSA 1024 bits en mode PKCS1V1.5 ayant comme algorithme de hash le SHA256 .

La basiccard : carte sur laquelle des applications peuvent être développées en basic. La carte est munie d'un processeur cryptographique offrant le chiffrement, le hash et le RSA

3.1 Questions Diverses (deux lignes par réponse)

- Pour quelles raisons il est intéressant d'utiliser la fonction hash de la carte ?
- Est-il obligatoire d'utiliser la fonction hash de la carte ?
- Les cartes possèdent la fonction de génération de clés privées RSA. Pour quelles raisons est-il intéressant d'utiliser cette fonction ?

3.2 Choix

La famille contient plusieurs types de cartes

Carte	Algorithme Asymétrique	Chiffrement	Hash	Système Fichiers
ZC4.5A	RSA 1024 bits	AES	SHA1	Oui
ZC4.5D	RSA 1024 bits	DES	SHA1	Oui
ZC5.4	Courbes Elliptiques 211	AES/DES	SHA256	Oui

- Question : Quelle carte vous semble la mieux adaptée pour les besoins de l'exercice ? Justifier (4 lignes)

3.3 Implémentation

Les fonctionnalités RSA sont faites par les routines système suivantes:

RsaPKCS1Sign (*Hash$, p$, q$, e$, Sig$***)**
RsaPKCS1Verify (*Hash$, n$, e$, Sig$***)**
RsaEncrypt (*Mess$, n$, e$***)**
RsaDecrypt (*Mess$, p$, q$, e$***)**

This procedure computes *Mess$ e$* modulo *n$*, returning the result in *Mess$*.
Les specifications des routines sont les suivantes:

As described in **PKCS #1 v2.0: RSA Cryptography Standard**, a *signature scheme with appendix* consists of a *signature generation operation* and a *signature verification operation*. One signature scheme with appendix is defined: **RSASSA-PKCS1-v1_5**.

The **RSA** Plug-In Library uses **SHA-1** as the hash function for the signature scheme.
To generate a signature using the **RSASSA-PKCS1-V1_5-SIGN** signature generation operation:
Call RsaPKCS1Sign (*Hash$, p$, q$, e$, Sig$***)**
Hash$ The 20-byte **SHA-1** hash of the data to be signed.
p$, q$, e$ The private key (*p*, *q*, *e*).
Sig$ The signature calculated by **RsaPKCS1Sign**. It has the same size as *n$* (where *n* = *pq* is the public-key modulus).

To verify a signature using the **RSASSA-PKCS1-V1_5-VERIFY** signature verification operation:
SignatureValid = **RsaPKCS1Verify (***Hash$, n$, e$, Sig$***)**
Hash$ The 20-byte **SHA-1** hash of the data that was signed.
n$, e$ The public key (*n*, *e*).
Sig$ The signature to be verified.
SignatureValid **True** if the signature is valid.

Call RsaEncrypt (*Mess$, n$, e$***)**
This procedure computes *Mess$ exp(e$)* modulo *n$*, returning the result in *Mess$*.

Call RsaDecrypt (*Mess$, p$, q$, e$***)**
This procedure first computes *d$* = inverse of *e$* modulo (*p$*-1)(*q$*-1). Then it computes *Mess$ exp(d$)* Modulo(*p$ q$)*, returning the result in *Mess$*.

- Comment arriver à implémenter le RSA avec SHA256 à partir des routines précédentes ?
- Si une application veut faire le RSA avec un autre type de padding que le PKCS1V1.5 pourra-t-elle le faire ? Justifier ?

4. PKCS10

Il est conseillé de bien partitionner la requête et d'être clair dans son implémentation.
Soit la définition ASN1 de demande de certification

```
CertificationRequestInfo ::= SEQUENCE {
   version      INTEGER { v1(0) } (v1,...),
   subject      Name,
   subjectPKInfo SubjectPublicKeyInfo
  }

SubjectPublicKeyInfo  ::= SEQUENCE {
   algorithm            AlgorithmIdentifier,
   subjectPublicKey     RSAPublicKey
}
CertificationRequest ::= SEQUENCE {
   certificationRequestInfo CertificationRequestInfo,
   signatureAlgorithm       AlgorithmIdentifier,
   signature            BIT STRING
}
```

```
AlgorithmIdentifier  ::= SEQUENCE {
   algorithm       id,
   parameters    Type OPTIONAL
}

Name ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::= SET OF AttributeTypeValue

AttributeTypeValue ::= SEQUENCE
{
  type             OBJECT IDENTIFIER,
  value             ANY
}

RSAPublicKey ::= SEQUENCE {
  modulus BIT STRING,   (valeur positive)
  publicExponent BIT STRING (valeur positive)
}
```

4.1 Questions diverses

- Dans quelle partie du PKCS une clé publique RSA est définie ?
- Dans quelle partie du PKCS les attributs du DistinguishedName sont définis ?
- Comment une autorité de certification peut authentifier l'émetteur d'une requête PKCS10 pour un certificat de type messagerie ?
- Pour quelle raison CertificationRequest comporte une signature ?

4.2 Implémentation DER

La requête PKCS10 est faite pour un certificat de type messagerie.
Implémenter la requête PKCS10 en DER en prenant pour valeurs particulières :
L'email est: toto.dupont@nowhere.com
La clé publique est : 0x01 00 01 (exposant), AA BB CC DD …. 01 24 37 (modulo sur 1024 bits)
L'OID de l'algorithme est le RSA_SHA1 : 1.2.840.113549.1.1.5
La signature : 0x7F DE 01 23 45 …. AB C1

5. Question Bonus

Soit à tester un nombre aléatoire x s'il est premier. Il est testé tout d'abord s'il est divisible par l'un des 100 premiers nombres premiers Pi; s'il ne l'est pas il est testé s'il est premier avec 20 nombres aléatoires impairs. Si x a un diviseur parmi les 100 petits nombres premiers, on fait x = x +2 puis on répète les tests.

- Rappel : x a un diviseur Pi ⇔ x = 0 mod (Pi).
- Si x = r mod (Pi), que vaut x+2 mod(Pi) ?
- Quelle est la taille maximale en octets de x mod(Pi) ?
- Ecrire la formule logique : « x n'a aucun diviseur parmi les 100 premiers nombres premiers Pi »
- Ecrire la formule logique : « x a au moins un diviseur parmi les 100 premiers nombres premiers Pi »
- Comment implémenter d'une façon rapide la génération d'un nombre x qui n'a pas de diviseur parmi les 100 premiers nombres premiers ? (Méthode du Tableau de Restes).