Composition de Cryptographie - 2004

La composition est décomposée en 5 exercices de difficulté croissante.

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

Le support de cours est permis. La majeure partie des questions font appel au sens pratique d'ingénieur et le suivi du cours en classe, il est recommandé à l'élève de ne pas perdre son temps pour rechercher les solutions dans le support de cours.

1. Générateur de Nombres Aléatoires (4 points)

Un générateur de nombre aléatoire est utilisé pour aléatoirement (avec remise) des cartes dans un jeu de 52 cartes.

a) Quelle est la façon la plus simple de présenter une carte ?
b) Adapter le générateur range_rand (voir support de cours) pour générer des unsigned char sous la forme d'une routine char_range_rand.
c) Améliorer l'implémentation de char_range_rand pour la rendre :
   - équiprobable pour toutes les valeurs
   - et la plus rapide possible.

2. Protection d'un fichier par une passphrase (4 points)

Pour protéger un fichier par une passphrase (mot de passe sous forme d'une phrase) on utilise les techniques suivantes :

- chiffrement du fichier en AES mode CBC le padding suivant :0,0,0,…L avec L est la taille en octets du dernier bloc du fichier chiffré en AES, ou L=0 si le dernier bloc a la taille d'un bloc AES.
- La clé K de chiffrement AES = hash (passpharse)
- IV est fixe = 0000000000000..000

a) Quels algorithmes HASH sont adaptés pour cette protection et pourquoi ?
b) Quelle est la valeur maximale de L et pourquoi ?
c) comment vérifier la passphrase pour permettre l'accès au fichier?

3. Analyse d'une implémentation Crypto API (5 points)

Le but de cet exercice est d'analyser une implémentation CryptoAPI.

a) Indiquer pour les phases 0,2,3,4,5,6 les fonctionnalités implémentées.
b) A quoi sert les fonctionnalités entre les commentaires ''PHASE XY'' et ''FIN PHASE XY''
c) A quoi sert la fonction implémentée ?

```
int XXXX(pSecurityCntxtEx pDevice,uint lgCert, uchar *pCert,
                        int  algoid, int mode, uint lgHash, uchar *Hash,uint lgSignature, uchar *Signature,
                        uint *lgAnswer, uchar *Answer)
{
int phase = 0;
int cr = 0;
PCCERT_CONTEXT pCertCntxt = NULL;
HCRYPTKEY hk = 0;
HCRYPTHASH hHash=0;
char buffer[2048];
int i;

        while((phase <7) && (cr==0))
        {
                switch (phase)
                {
                case 0: //PHASE 0
                        *lgAnswer = 0;
                        pCertCntxt =    CertCreateCertificateContext(MY_TYPE,  pCert,  lgCert);
                        if(pCertCntxt==NULL) cr = e_badkeyopen;
                        break;
```

```
                case 1: // Obtention d'un contexte de clé publique
                        cr = capi_PKfromCert((HCRYPTPROV)pDevice->handle, &hk,pCertCntxt);
                        break;
                case 2: // PHASE 2
                        if(!CryptCreateHash((HCRYPTPROV)pDevice->handle, algoid,  0,0,&hHash))
                                cr = e_badexec;
                        break;
                case 3: // PHASE 3
                        if(!CryptSetHashParam(hHash, HP_HASHVAL, Hash, 0 ))
                                                cr  = e_badparam;
                        break;

                case 4: // PHASE 4
                        for (i=0; i<lgSignature; i++)  buffer[i] = Signature[lgSignature-i-1];
                        if(!CryptVerifySignature( hHash, buffer, lgSignature, hk,0, 0))
                                        cr  = e_badsignature;
                        break;
                case 5: // PHASE 5
                         if(!CryptGetHashParam( hHash, HP_HASHVAL, buffer, &lgSignature, 0 ))
                                        cr = e_badparam;
                         break;
                case 6: // PHASE 6
                        *lgAnswer = lgSignature;
                        if(Answer!=NULL) memcpy(Answer, buffer, lgSignature);
                        break;

                default: break;

                }
                phase++;
        }
// PHASE XY
        if (hk!=(HCRYPTKEY)NULL)  CryptDestroyKey(hk);
        if (hHash!=0)  CryptDestroyHash(hHash );
        if(pCertCntxt!=0)  CertFreeCertificateContext(pCertCntxt);
// FIN PHASE XY

        return(cr);
}
```

4. API de sécurité PKCS11 (7 points)

Le but de cet exercice est d'implémenter les services de non répudiation en utilisant les requêtes de sécurité PKCS11. Soit un fichier qu'il faut transmettre d'un client à sa banque en clair. Pour chaque routine PKCS11 utilisée un commentaire doit être ajouté pour indiquer la technique de sécurité utilisée.

a) Implémenter la signature en utilisant les requêtes de sécurité PKCS11. (ne pas inclure les requêtes d'accès au dispositif)
b) Implémenter la vérification signature en utilisant les requêtes de sécurité PKCS1. (ne pas inclure les requêtes d'accès au dispositif ni la vérification de la validité des certificats).

5. Obtention des composants premiers p et q à partir de S, N et P (5 points)

Soit une bi-clé RSA dont on connaît les exposants S et P ainsi que le modulo N.

a) A partir des équations de base de la génération d'une bi-clé RSA écrire les relations liant p et q à S,N,P en remplaçant la fonction mod par sa définition : $a = b \bmod c \Leftrightarrow \exists k$ tq. $a = k * c + b$

b) Démontrer que ces relations peuvent se réduire à une équation du second degré.

c) Comment choisir k ou quelles sont les conditions pour résoudre les équations ?