Principe et Algorithme Cryptographiques : DS n°2
première session

Université de Lille-1
FIL — PAC

20 mai 2016

Ce sujet est composé de 3 pages. Vos réponses doivent **absolument** être justifiées. Les différentes parties sont indépendantes et ne sont pas classées par ordre de difficulté. Toutes contiennent des questions de difficultés variés.

1 Faille d'OpenSSL dans Debian

Voici à peu près ce qui se passe quand on invoque la bibliothèque OpenSSL en réclamant la production d'une paire de clefs RSA :

1. Un générateur pseudo-aléatoire de qualité cryptographique est initialisé de manière imprévisible.
2. Deux nombres premiers aléatoires, p et q sont générés.
3. La clef publique (n, e) et la clef secrète d sont produites à partir de p et q.

À toute fin utile, on peut imaginer que la procédure de génération d'un nombre premier aléatoire de k bits fonctionne de la façon suivante :
— Générer un nombre (impair) de k bits uniformément au hasard
— Tant qu'il n'est pas premier, revenir à l'étape précédente.

▷ **Question 1 :** Quel est l'ordre de grandeur du temps nécessaire à la production d'un nombre premer aléatoire de 1000 bits (en 2016, sur une machine normale) ?

▷ **Question 2 :** En gros, à combien d'itérations de la procédure peut-on s'attendre avant d'obtenir le résultat ?

▷ **Question 3 :** Quel est l'ordre de grandeur du nombre de bits pseudo-aléatoires nécessaires pour produire une paire de clefs RSA de 2048 bits par cette procédure ?

En septembre 2006, un programmeur de la distribution Debian de Linux a modifié le code source d'OpenSSL pour éviter un avertissement lors de l'exécution d'un analyseur statique. Une conséquence inattendue de la modification, repérée en mai 2008, est que le générateur pseudo-aléatoire n'est initialisé qu'avec le `PID` du processus courant (sur Linux, c'est un nombre compris entre 0 et 2^{15}).

▷ **Question 4 :** J'ai généré une paire de clefs RSA avec la version 2006–2008 d'OpenSSL, et j'ai posté ma clef publique sur internet. Expliquer comment un adversaire pourrait tirer parti de la modification d'OpenSSL pour obtenir ma clef secrète.

2 Faiblesse de la fonction `rand()` en C

Cet exercice est une des missions « secrètes » du TP, que personne n'a fait. Considérons le programme suivant, en langage C.

```c
#include <stdint.h>
#include <stdio.h>

void encrypt(char *message, int message_size) {
  uint32_t K[4], IV[2];
  FILE *output;

  srand48( time(NULL) );
```

```
  K[0] = mrand48();
  K[1] = mrand48();
  K[2] = mrand48();
  K[3] = mrand48();

  IV[0] = mrand48();
  IV[1] = mrand48();
  IV[2] = mrand48();
  IV[3] = mrand48();

  printf("Randomly generated key : %x-%x-%x-%x\n', K[0], K[1], K[2], K[0])

  ciphertext = aes_128_CBC(plaintext, message_size, K, IV);

  output = fopen("ciphertext.bin", "w");
  fwrite(IV, 16, 1, output);
  fwrite(ciphertext, message_size, 1, output);
  fclose(output);
}
```

En gros, c'est une fonction qui génère une clef symétrique aléatoire, ainsi qu'un IV aléatoire. La clef est affichée (nous supposons qu'elle est écrite sur un bout de papier par l'utilisateur). Le message est chiffré puis écrit dans un fichier, précédé par l'IV.

Le générateur pseudo-aléatoire utilisé, la fonction `mrand48()`, fait partie de la norme POSIX. Elle doit être implémentée de la façon suivante :

```
1  uint64_t rand48_state;
2
3  void srand48(uint32_t seed) {
4    rand48_state = 0x330e + (seed << 16);
5  }
6
7  uint32_t mrand48() {
8    rand48_state = (0x00000005deece66d * rand48_state + 11) & 0x0000ffffffffffff;
9    return (rand48_state >> 16);
10 }
```

L'appel à `srand48()` permet d'entrer une « graine » dans le générateur pseudo-aléatoire. Chaque appel à `mrand48()` renvoie 32 bits pseudo-aléatoires, et met à jour l'état interne de la fonction. Le générateur tire son nom du fait que son état interne occupe 48 bits (en effet, les 16 bits de poids fort de la variable `rand48_state` sont toujours nuls).

▷ **Question 5 :** Cette manière d'initialiser le générateur pseudo-aléatoire vous semble-t-elle raisonnable ?

▷ **Question 6 :** Décrivez une attaque très simple qui permet de déchiffrer le fichier `ciphertext.bin` en 2^{32} essais (sans connaître la clef, bien sûr).

▷ **Question 7 :** En terme d'arithmétique modulaire, quel est l'effet de l'opération AND de la ligne 8 ?

▷ **Question 8 :** Supposons qu'on connaisse la valeur de la variable `rand48_state`, juste après l'exécution de la ligne 8 dans la fonction `mrand48()`. Expliquez comment on peut faire pour « remonter » efficacement à la valeur qu'avait cette variable *avant* l'exécution de la ligne 8.

▷ **Question 9 :** Lorsqu'on connaît le premier mot de 32 bits de l'IV, connaît-on entièrement l'état interne du générateur pseudo-aléatoire ? Sinon, que manque-t-il ?

▷ **Question 10 :** Déduisez-en une attaque qui permet de déchiffrer le fichier `ciphertext.bin` en 2^{16} essais. Ceci est-il réalisable en pratique, sans moyens spéciaux ?

3 Protocole « Secure Remote Password »

Le protocole « Secure Remote Password »(SRP) est un protocole d'échange de clef authentifié par mot de passe conçu dans les années 2000. Il s'agit d'une variante de l'échange de clef Diffie-Hellman. Par conséquent, on note p un grand nombre premier utilisé dans le protocole, et g un générateur raisonnable modulo p.

▷ **Question 11 :** Rappelez ce que sont les « PAKE »(Password Authenticated Key Exchange). Quelle type d'attaque visent-ils en particulier à éviter ?

La caractéristique la plus frappante du protocole SRP, c'est que le serveur ne connaît pas le mot passe — et pourtant il peut vérifier que le client le connaît.

▷ **Question 12 :** Quel avantage est-ce que ceci procure par rapport à un protocole tel que SPEKE où le serveur possède le mot de passe ?

Le protocole est composé de deux procédures : l'*enregistrement* (où un utilisateur s'inscrit sur le serveur) et la *connection* (où un utilisateur préalablement inscrit se connecte au serveur).

Enregistrement Pour s'enregistrer auprès du serveur, le client, dont l'identité est une chaine de bits I, et dont le mot de passe est une (courte) chaine de bits pwd, doit :

1. Calculer $x \leftarrow H(pwd)$
2. Calculer $v \leftarrow g^x \bmod p$ (c'est le « vérifieur »).
3. Envoyer (I, v) au serveur, qui les stocke.

▷ **Question 13 :** Le serveur apprend-il le mot de passe au terme de la procédure d'enregistrement ?

▷ **Question 14 :** Supposons qu'un attaquant parvienne à intercepter la transmission pendant la phase d'enregistrement. Expliquez comment il pourrait s'y prendre pour tenter de découvrir le mot de passe.

Dans toute la suite, on suppose que la procédure d'enregistrement des utilisateurs est effectuée sur un canal sécurisé (par exemple, en accédant physiquement au serveur).

Connection La phase de connexion fonctionne de la façon suivante.

1. Le client génère un nombre aléatoire a, calcule $A \leftarrow g^a \bmod p$, puis envoie (I, A) au serveur (celui-ci interrompt le protocole si $A \equiv 0 \bmod p$).
2. Le serveur récupère le vérifieur v associé à l'identité I dans sa base de donnée. Il choisir un nombre aléatoire b, calcule $B \leftarrow v + g^b \bmod p$. Enfin, il envoie B au client (celui-ci interrompt le protocole si $B \equiv 0 \bmod p$).
3. Le client et le serveur calculent $u \leftarrow H(A \parallel B)$.
4. Le client calcule $S \leftarrow (B - g^x)^{a+ux} \bmod p$, puis $K \leftarrow H(S)$.
5. Le serveur calcule $S' \leftarrow (A \cdot v^u)^b \bmod p$, puis $K' \leftarrow H(S)$.
6. Le client calcule $M_1 \leftarrow H(I \| A \| B \| K)$, puis l'envoie au serveur (celui-ci interrompt le protocole si la valeur fournie est incorrecte).
7. Le serveur calcule $M_2 \leftarrow H(A \| M_1 \| K')$, puis l'envoie au client (celui-ci interrompt le protocole si la valeur fournie est incorrecte).

▷ **Question 15 :** Montrer que si les deux participants sont honnêtes, alors $S = S' = \left(g^b\right)^{a+ux}$.

▷ **Question 16 :** Quel est l'intérêt des deux dernières étapes ?

▷ **Question 17 :** La valeur de u peut-elle être apprise par un adversaire passif ?

▷ **Question 18 :** Expliquez pourquoi un attaquant qui espionnerait une exécution du protocole ne pourrait pas monter une attaque contre le mot de passe.

▷ **Question 19 :** Alice exécute le protocole avec un serveur. Charlie connaît le mot de passe d'Alice (ils sont intimes), et en plus il espionne l'exécution du protocole. Peut-il obtenir la clef de session K ?

▷ **Question 20 :** Un adversaire passif qui ne connait pas le mot de passe peut-il apprendre la clef de session K ?