TP sécurité FREAK + LogJam

October 11, 2023

1 Rappel

Une clé privée RSA est calculée de la manière suivante

$$e \times p \equiv 1 \pmod{\varphi(n)}$$

- **e** fait partie de la clé publique : on le connaît
- **n** fait partie de la clé publique : on le connaît
- $\boldsymbol{\varphi(n) = (p-1)(q-1)}$
- $\boldsymbol{n = p \times q}$
- **p** et **q** sont des nombres premiers que nous ne connaissons pas

Un message chiffré m chiffré avec une clé publique n peut être retrouvé grâce à sa clé privée avec

$$m^d \pmod{n}$$

Donc :

1. Pour trouver **d**, il faut trouver $\boldsymbol{\varphi(n)}$
2. Pour trouver $\boldsymbol{\varphi(n)}$ il faut trouver **p** et **q**
3. Pour trouver **p** et **q** il faut résoudre le problème de factorisation en nombre premiers de **n**.

2 Exercice 1 - Déchiffrement de clé RSA

La faille principale des attaques FREAK et LogJam vient de la faciliter de déchiffrer des clés RSA de 512 bits. Dans cette exercice, nous allons créer un algorithme de déchiffrement de clé RSA afin de pouvoir constater la faciliter de création de ceux-ci et ainsi déchiffrer un message qui aurait pu être intercepté par un attaquant

2.1 Partie 1 - Création de l'algorithme

La méthode de factorisation de Fermat est un algorithme de décomposition en produit de facteurs premiers d'un entier naturel. Implémentez cette méthode dans un programme Python qui prendra n en entrée et qui retourne p et q. La méthode correspond au pseudo-code suivant :

Algorithm 1 FactorFermat(n)

$A \leftarrow \lceil \sqrt{N} \rceil$
$B \leftarrow A \times A - N$
while $\sqrt{B} \notin \mathbb{N}$ **do**
  $A \leftarrow A + 1$
  $B \leftarrow A \times A - N$
end while
$p \leftarrow A + \sqrt{B}$
$q \leftarrow A - \sqrt{B}$
return p, q

2.2 Partie 2 - vérification de l'efficacité de l'algorithme

2.2.1 Test avec un n aléatoire

Exécuter votre algorithme avec $n = 55$

2.2.2 Test avec le n d'une clé publique

Vous avez dans le dépot git une clé publique que vous allez essayer de déchiffrer afin de récupérer la clé privée liée et ainsi déchiffrer le message chiffré lui aussi présent sur le dépot git

Quelque conseil :

1. ```
   $ openssl rsa -pubin -in public_key.pem -text -noout
   ```

   permet de récupérer des informations sur votre clé publique

2. Vous pouvez utiliser l'interpréteur python3 pour déchiffrer une nombre héxadécimal à l'aide du code

   ```
   decimal_value = int(hex_value, 16)
   ```

Vous allez ainsi pouvoir récupérer un nouveau n que vous pourrez alors utiliser dans votre algorithme afin de récupérer p et q

2.2.3 Analyse

Que remarquez-vous, que pensez vous qu'il se passerait si la clé public était de 1046 bits comme l'oblige maintenant les serveurs web ?
   ```

2.3 Déchiffrer le message

Ce code n'arrivera jamais à sa fin à cause des faibles capacités de python lors de calcul avec des grands nombres. Pour que vous puissiez quand même avancer sur le TP nous vous avons donc fourni la clé privé utilisée pour chiffrer le message.

2.3.1 Trouver d

Vous pouvez récupérer des informations sur cette clé avec la commande suivante :

```
$ openssl rsa -in private_key.pem -text -noout
```

Parmis toute ces informations, trouvez celle qui pourrait vous servir à trouver d, la clé de chiffrement de votre message.

2.3.2 Decrypt

Mettez en place la fonction **Decrypt(message, d, n)** qui va vous permettre de déchiffrer le message

Une fois le message déchiffré, vous verez une suite de caractère aléatoire suivit du message. Cela est du au padding automatique fait par le chiffrement RSA afin de faire en sorte que tout les messages ai la même longueur. Pas de panique, vous avez réussi

2.4 À savoir

Ici, nous n'avons pas ou vous faire calculer directement p et q à cause des faibles capaciter de python mais les fonctions permettantes de trouver d avec ces deux variable sont très facile à mettre en place et il serait en faite possible de récupérer la clé privé à partir de la clé publique en quelque seconde dans de meilleurs conditions. Par contre, il est presque impossible de le faire pour des clés RSA de plus de 512 bits car le temps de résolution de l'algorithme de factorisation de Fermat deviens bien trop long

3 Exercice 2 - Diffie-Hellman

Le chiffrement Diffie-Hellman est un protocole de sécurité utilisé pour permettre à deux parties de s'échanger des clés de manière sécurisée sur un canal de communication non sécurisé. Il repose sur le concept mathématique de la "difficulté du problème du logarithme discret". Les deux parties, Alice et Bob, choisissent secrètement des nombres et effectuent des calculs pour générer une clé de session commune sans jamais échanger cette clé directement. Ainsi, même si un attaquant intercepte les échanges, il est extrêmement difficile de déterminer la clé de session sans les informations secrètes d'Alice et de Bob. Ce protocole est fondamental dans la sécurité des communications en ligne.

3.1 Calcul des clés

Un code python à compléter vous est founi dans le répos codefirst, à vous de le compléter